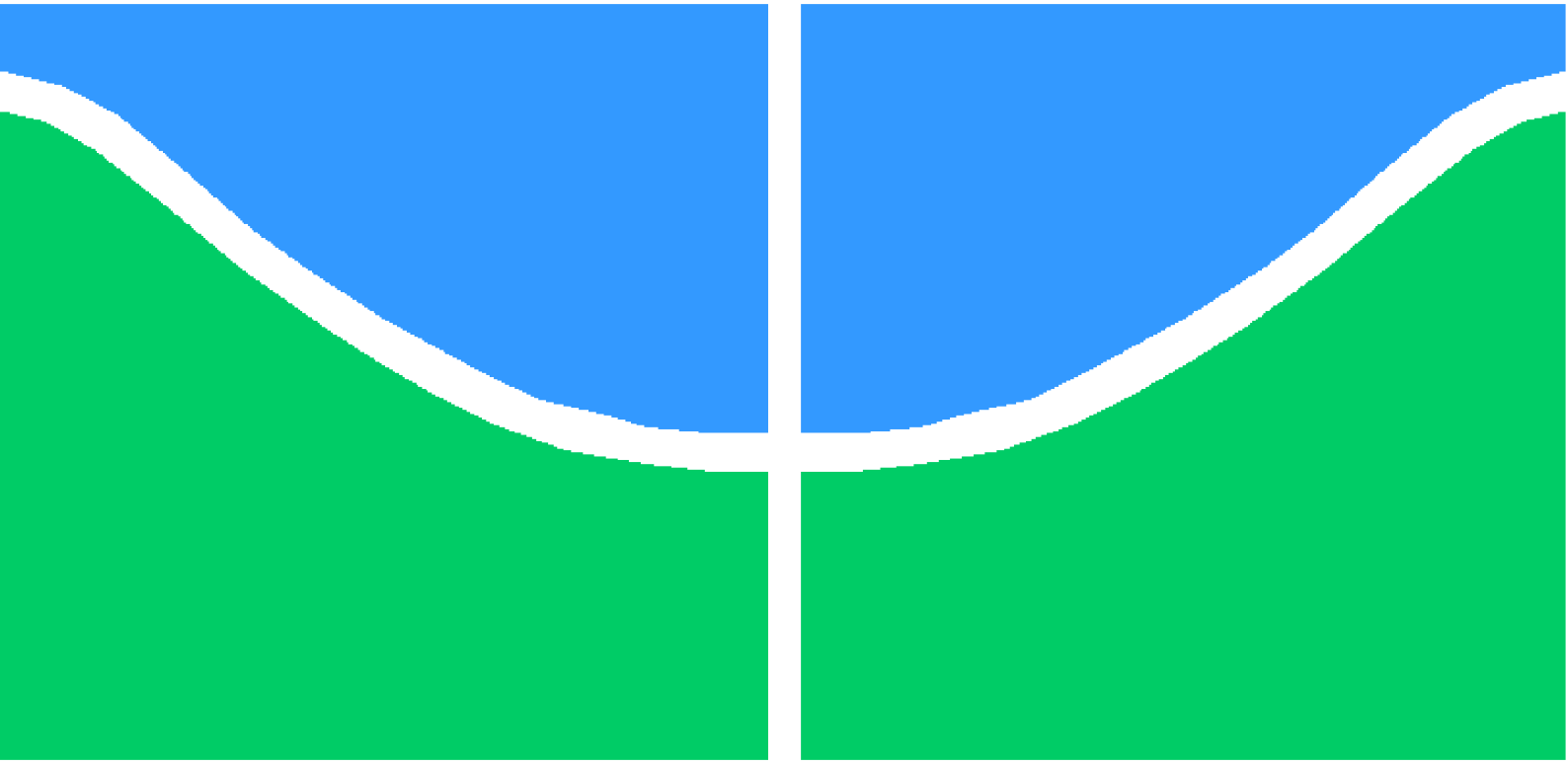

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Painel de acompanhamento de medidas e indicadores para monitoramento da qualidade de software

Autor: Adriano de Jesus Barboza

Orientador: Dr. Luiz Carlos Miyadaira Ribeiro Júnior

Brasília, DF

2013

Adriano de Jesus Barboza

Painel de acompanhamento de medidas e indicadores para monitoramento da qualidade de software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Luiz Carlos Miyadaira Ribeiro Júnior

Brasília, DF

2013

Adriano de Jesus Barboza
Painel de acompanhamento de medidas e indicadores para monitoramento da qualidade de software/ Adriano de Jesus Barboza. – Brasília, DF, 2013-
77 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Luiz Carlos Miyadaira Ribeiro Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2013.

1. Indicadores. 2. Qualidade. I. Dr. Luiz Carlos Miyadaira Ribeiro Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Painel de acompanhamento de medidas e indicadores para monitoramento da qualidade de software

CDU 02:141:005.6

Adriano de Jesus Barboza

Painel de acompanhamento de medidas e indicadores para monitoramento da qualidade de software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 01 de junho de 2013:

Dr. Luiz Carlos Miyadaira Ribeiro Júnior
Orientador

Msc. Cristiane Soares Ramos
Convidado 1

Msc. Ricardo Ajax Dias Kosloski
Convidado 2

Brasília, DF
2013

Este trabalho é dedicado aos meus pais, Oséas e Piedade, que me ensinaram, dentre tantas outras coisas, que não devo desistir em qualquer dificuldade.

Agradecimentos

Primeiramente a Deus por ter me confiado essa oportunidade, por ter me dado forças nos momentos difíceis, e por ter agido através de meus pais e amigos, para que me ajudassem a ter ânimo para chegar ao fim dessa etapa.

Aos meus pais pelos incentivos que deram aos meus estudos desde que era pequeno, pelo investimento feito em minha educação, pelo ânimo injetado em mim nos momentos de dificuldades, pela paciência que tiveram comigo durante esse processo, pelas ajudas que me deram durante essa etapa, pela confiança, pelo amor e pelo carinho.

Aos meus amigos e familiares que me deram forças, palavras de incentivos e acreditaram em mim.

Aos meus amigos da faculdade que por diversas vezes foram meus apoios nessa caminhada, que me incentivaram e me ajudaram a chegar aqui.

Aos meus professores que se dedicaram em passar o conhecimento necessário à minha formação durante todo esse processo.

Enfim, sem vocês, não poderia ter concluído essa etapa.

"Bendito o homem que deposita a confiança no Senhor,
e cuja esperança é o Senhor.
(Bíblia Sagrada, Jeremias 17, 7)

Resumo

Qualidade de Software tem sido uma preocupação crescente no contexto de desenvolvimento e manutenção de software. Para que software de qualidade seja desenvolvido, é necessário gerenciar o projeto da maneira adequada. Entretanto, gerenciar um projeto não é uma tarefa simples, principalmente devido ao caráter abstrato que o software apresenta, o que dificulta a visibilidade sobre alguns aspectos do projeto, incluindo a Qualidade. Contudo, medidas e indicadores podem auxiliar os gestores a obter maior visibilidade sobre o projeto. Nesse contexto, este trabalho propõe a construção de um painel de acompanhamento (dashboard) de medidas e indicadores relacionados a atividades de Teste de Software para o monitoramento da qualidade do produto ao longo do ciclo de vida de desenvolvimento e manutenção, a ser implementado em um ambiente real, o laboratório CQTS. Para que essas medidas e indicadores sejam coerentes ao contexto, será utilizada a abordagem GQM para a definição dessas medidas.

Palavras-chaves: qualidade. gerenciamento de projeto. testes de software. medidas. indicadores.

Abstract

Software Quality has been a growing concern in the context of developing and maintaining software. For quality software is developed, it is necessary to manage the project properly. However, managing a project is not a simple task, mainly due to the abstractness that the software has, thus hampering visibility on some aspects of the project, including Quality. However, measures and indicators can help managers gain visibility over project. In this context, this paper proposes the construction of a monitoring panel (dashboard) measures and indicators related to activities of Software Testing for monitoring product quality throughout the lifecycle of development and maintenance, to be implemented in a real environment, the laboratory CQTS. For these measures and indicators are consistent context, will be used the GQM approach to defining these measures.

Key-words: quality. project management. software testing. measures. indicators.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Este Capítulo apresenta, na Seção 1.1 serão indicados os fatores motivacionais para a realização deste trabalho, ressaltando sua relevância no contexto de desenvolvimento e manutenção de software. Finalmente na Seção 1.3 serão apresentados os objetivos que o presente trabalho visa alcançar.

1.1 Motivação

Para que software de qualidade seja entregue o projeto tem que ser gerenciado. Entretanto gerenciar um projeto pode não ser uma tarefa fácil e é preciso ter visibilidade sobre parâmetros do projeto. Contudo, muitas vezes, gestores de projetos de software não obtêm, facilmente, uma visão consolidada sobre tais parâmetros, pois software, ao contrário de outras engenharias, trata de produtos intangíveis (SOMMERVILLE, 2011). Dentre tais parâmetros de projeto, encontram-se medidas de prazo, custo e qualidade. Frequentemente, gerentes de projeto se baseiam nos dois primeiros, como taxa de entrega e valor agregado (Índice de Desempenho de Prazo e Índice de Desempenho de Custo). Relacionado a qualidade, algumas medidas frequentes são densidade de defeitos, quantidade de defeitos, defeitos por tipo e severidade (MONTEIRO, 2008). No entanto, percebe-se que tais parâmetros de qualidade não são contextualizados ao ciclo de vida de desenvolvimento, e muitas vezes não fornecem benefícios de monitoramento por serem coletadas tarde demais (SHERRIFF, 2005).

Tais medidas de qualidade podem ser geradas a partir das atividades de Testes de Software. Em uma abordagem mais moderna de desenvolvimento, Teste de Software deve ser visto como uma disciplina que perpassa todo o ciclo de desenvolvimento, na qual diferentes aspectos de qualidade são verificados e validados de acordo com o momento do ciclo de vida de desenvolvimento. É neste contexto que este trabalho se insere, no sentido de fornecer, a partir de parâmetros de qualidade de produto, uma visão clara sobre o andamento do projeto.

1.2 Problema

O problema identificado e atacado neste trabalho é a baixa visibilidade de parâmetros de qualidade do software ao gestor de projetos.

1.3 Objetivo

1.3.1 Objetivo Geral

O objetivo deste trabalho é a proposta de um painel de acompanhamento (*dashboard*) para gestores ou líderes de projetos de software, que contenha medidas e indicadores previamente levantados, relacionados ao contexto de desenvolvimento e manutenção de software. O intuito é que tais medidas e indicadores possam dar um *feedback* aos gestores, para que possam ter visibilidade do andamento das atividades de teste e correção de defeitos, fazendo um acompanhamento do estado atual do software, bem como possam ter insumos para entender e identificar possíveis fragilidades do processo e anormalidades do andamento do projeto.

1.3.2 Objetivos Específicos

- Definir medidas e indicadores coerentes ao contexto CQTS
- Configurar o ambiente de software necessário no laboratório CQTS
- Implementação do software que apresente tais medidas e indicadores anteriormente levantados.

2 Referencial Teórico

Este Capítulo apresentará uma visão dos principais temas que permeiam o trabalho desenvolvido. Está dividido em seções que dividem os diferentes temas, conforme explicado a seguir.

Na Seção 2.1 são apresentados conceitos de gerenciamento de projetos, trazendo também uma visão das dificuldades relacionadas. A Seção 2.2 traz uma visão sobre Qualidade de Software e sua importância. A Seção 2.3 ressalta a importância da gestão da qualidade no contexto de desenvolvimento de software. A Seção 2.4 apresenta a relevância das atividades de teste de software e a relação dessas atividades com a garantia de qualidade. Finalmente na Seção 2.5 são apresentados os principais conceitos relacionados à Medição e Análise bem como sua importância no auxílio da gestão da qualidade. Ainda na Seção 2.5 será apresentado a abordagem *Goal-Question-Metric* (GQM) para levantamento de medidas, e posteriormente serão apresentadas algumas medidas de software encontradas na bibliografia.

2.1 Gerenciamento de Projetos

Segundo o PMBOK (2008) "um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo". Difere de serviços continuados, que compreendem operações contínuas e repetitivas, entretanto possuem características em comum: executados por pessoas; restringidos por recursos limitados; planejados, executados e controlados. Ainda segundo o PMBOK (2008) os projetos são, frequentemente, componentes críticos da estratégia de negócios das organizações.

Nesse contexto, gerenciamento de projetos é a aplicação de conhecimentos, habilidades, ferramentas e técnicas às atividades do projeto a fim de atender seus requisitos (PMBOK, 2008). Gerenciamento de projetos de software é uma área essencial da Engenharia de Software. Projetos têm que ser gerenciados, pois, normalmente, a engenharia de software está sujeita a orçamentos e restrições de prazos organizacionais, e o papel do gestor é assegurar que o projeto de software atenda e supere essas restrições, além de entregar software de qualidade (SOMMERVILLE, 2011). O gerenciamento de projetos é importante, pois, "O bom gerenciamento pode não garantir o sucesso do projeto. Entretanto, um mau gerenciamento normalmente resulta no fracasso do projeto: software entregue atrasado, custando mais do que o previsto, ou sem atender as expectativas dos clientes" (SOMMERVILLE, 2011). Gerenciar um projeto pode ser uma tarefa árdua, pois

a construção e manutenção de grandes projetos de software é um processo complexo e sujeito a erros (PAUL et al., 1999).

Segundo Sommerville (2011) os critérios de sucesso de gerenciamento de projetos variam de acordo com as características de cada projeto, entretanto, observa-se na maioria dos projetos objetivos importantes como entregar o software para o cliente no tempo acordado, manter os custos dentro do orçamento, entregar software que atenda as expectativas do cliente, manter a equipe motivada e funcionando bem. Esses objetivos não são exclusivos da engenharia de software, mas também das outras engenharias. Entretanto o gerenciamento de projetos de software é um tanto desafiador, pois contam com, dentre outras, algumas diferenças:

1. Produto intangível: o gestor de uma construção de um prédio ou um engenheiro civil podem ver claramente evidências sobre o andamento da construção, as áreas que não estão terminadas, as que já estão, pois um prédio é algo tangível, ao contrário do software que não pode ser visto dessa maneira, que não pode ser tocado.

2. O gerenciamento de projetos de software de grande porte se diferencia de demais tipos de projeto em alguns aspectos, pois, mesmo que os gestores tenham boa bagagem de experiência ainda é difícil prever problemas, pois as rápidas mudanças tecnológicas podem tornar essa bagagem de experiência obsoleta, fazendo com que as experiências obtidas de projetos anteriores possam não ser transferíveis para os novos projetos.

3. Processos de software são variáveis e de acordo com as organizações: alguns processos de engenharia para alguns sistemas como pontes e edifícios são bem compreendidos. No entanto, os processos de software variam consideravelmente de uma organização para outra, mesmo havendo progressos significativos na padronização de processos.

2.2 Qualidade de Software

Gerenciar um projeto inclui atenção em diversos fatores, dentre eles a qualidade (PMBOK, 2008). É comum observar na indústria de produtos não abstratos, como a de carros, a produção de vários produtos exatamente iguais, ao seguirem um mesmo processo de construção. Entretanto, software, por ser abstrato, produzirá provavelmente produtos distintos mesmo que se siga o mesmo processo. Além disso, a indústria de software em comparação às outras existentes é relativamente recente (LEAL, 2008).

Problemas com qualidade de software foram inicialmente descobertos nos anos 60 com o desenvolvimento dos primeiros grandes sistemas de software (SOMMERVILLE, 2011). Softwares vêm sendo largamente utilizados nas mais diversas áreas e a procura pela qualidade do sistema contratado tem sido assim uma preocupação crescente. Tendo em

vista também a crescente complexidade dos sistemas desenvolvidos e crescente concorrência entre empresas fornecedoras, e o importante espaço que a indústria de software vem ocupando na economia mundial é fundamental que haja um foco no desenvolvimento de produtos de qualidade, que está relacionada ao grau de satisfação dos clientes em relação a um determinado produto (LEAL, 2008). Similarmente outro autor afirma que "Produção de software de alta qualidade é altamente necessário para realizar a satisfação absoluta do cliente na indústria de software" (NAIR; SUMA; TIWARI, 2012).

Segundo a NBR/ISO 9000 (2005) qualidade é o grau no qual um conjunto de características inerentes satisfaz aos requisitos, que são necessidades geradas pelos clientes ou por outras partes interessadas. Apesar de existir uma atualização da norma ISO/IEC 9126 (2001), um estudo realizado em 2009 indica que ainda é a norma mais utilizada entre as organizações no que diz respeito a requisitos de qualidade de software (SOUSA; MARINHO, 2009). Tal ISO separa a definição de qualidade em três dimensões: qualidade interna, qualidade externa e qualidade em uso. Define qualidade interna como "a totalidade das características do produto de software do ponto de vista interno." (ISO/IEC 9126, 2001). Por sua vez, a qualidade externa é definida como "a totalidade das características do produto de software do ponto de vista externo. É a qualidade quando o software é executado, o qual é tipicamente medido e avaliado enquanto está sendo testado num ambiente simulado, com dados simulados e usando métricas externas." (ISO/IEC 9126, 2001). Por sua vez, a qualidade em uso é "a visão da qualidade do produto de software do ponto de vista do usuário, quando este produto é usado em um ambiente e um contexto de uso especificados. Ela mede o quanto usuários podem atingir seus objetivos num determinado ambiente e não as propriedades do software em si" (ISO/IEC 9126, 2001).

2.3 Gestão de Qualidade

Gestão de qualidade é, a nível organizacional, um conjunto de processos e normas que busquem levar à produção de software de alta qualidade (SOMMERVILLE, 2011). A NBR/ISO 9000 (2005) traz uma definição semelhante quando diz que gestão da qualidade são atividades coordenadas para dirigir e controlar uma organização, no que diz respeito à qualidade. Ainda segundo a NBR/ISO 9000 (2005) controle de qualidade é o setor ou processo parte da gestão de qualidade que, com os requisitos em mãos, irá validar o produto que está sendo entregue ao cliente para analisar se está conforme com tais requisitos.

A gestão de qualidade reduz os custos de produção, pois, quanto mais cedo um defeito é localizado e corrigido, menos ele irá custar para ser reparado. E embora o investimento inicial possa ser substancial, o resultado em longo prazo será produtos de maior qualidade e custos de manutenção reduzidos (LEWIS, 2000). O custo total de uma gestão

eficaz de qualidade é a soma de quatro custos de componentes: prevenção, que consiste em ações tomadas para prevenir a ocorrência de erros; inspeção, que consiste em medir e avaliar produtos ou serviços de acordo com a conformidade com normas e especificações; custos de falhas internas, que são os incorridos na correção de defeitos antes do produto ser entregue; e custos de falhas externas que são os custos de defeitos descobertos depois que o produto foi lançado. Este último pode ser devastador, pois os resultados deles podem prejudicar o cliente e a reputação da organização (LEWIS, 2000).

2.4 Teste de Software

Conforme dito, premissa básica para que um software seja considerado de qualidade, é a adequação com seus requisitos. E para saber se um produto está de acordo com seus requisitos são realizadas atividades de teste de software. O teste de software é um elemento de um aspecto mais amplo, que é frequentemente referido como verificação e validação (V&V). V&V está diretamente ligado com a garantia de qualidade, que envolve ações tomadas para a redução de defeitos, pois sua definição engloba muita das atividades que são abrangidas por ela (PRESSMAN, 2006). Miller (apud Pressman (2006)) também relaciona teste de software com garantia de qualidade afirmando que "a motivação subjacente ao teste de programa é afirmar a qualidade de software com métodos que podem ser aplicados econômica e efetivamente tanto a sistemas de grande porte quanto de pequeno porte". Cabe aqui uma diferenciação entre defeito, erro e falha. Um defeito é definido como um desvio da especificação. Define-se que um sistema está em estado errôneo ou em erro se o processamento posterior a partir desse estado pode levar a defeito. Finalmente define-se falha (ou falta) como a causa física ou algorítmica do erro (WEBER, 2003).

Teste de produto é, portanto, mais relacionado com validação do que com verificação, sendo tradicionalmente considerado um processo de validação (LEWIS, 2000). Pressman (2006) similarmente afirma que verificação se refere ao conjunto de atividades que garante que o software implementa corretamente uma função específica, enquanto validação se refere ao conjunto de atividades que garante que o software construído corresponde aos requisitos do cliente. Boehm (1979) traz uma definição sucinta sobre a diferença entre as duas:

- 'Verificação: Estamos construindo corretamente o produto?'
- 'Validação: Estamos construindo o produto certo?'

O teste de software desempenha um papel extremamente importante em V&V, entretanto muitas outras atividades também são necessárias, que incluem revisões técnicas formais, estudo de viabilidade, revisão de documentação, revisão de base de dados,

dentre outras (PRESSMAN, 2006). Demais atividades, como essas, são necessárias, pois, tendo em vista que o teste de software é utilizado para averiguar se os requisitos foram alcançados, essa verificação é realizada no momento em que o produto, ou parte dele, já foi desenvolvido, o que pode ser, dependendo do momento, tarde para se incluir qualidade ao produto (LEWIS, 2000).

Tipicamente, um sistema de software comercial tem três estágios de testes (SOMMERVILLE, 2011):

1. Teste de desenvolvimento: quando o sistema é testado durante o desenvolvimento a fim de se descobrir defeitos. Designers de sistema e programadores são comumente envolvidos durante esse processo de teste.

2. Teste de release: quando um time separado de teste testa uma versão completa do software antes desta ser enviada para os usuários. O objetivo dessa etapa é testar se o sistema está atendendo às necessidades das partes envolvidas.

3. Teste por usuário: quando usuários, ou potenciais usuários do sistema, testam o sistema em seu próprio ambiente. Em alguns casos o usuário pode fazer parte da equipe interna para ajudar a decidir se o software pode ser comercializado, ou lançado uma release.

Uma das técnicas de testes é o desenvolvimento de casos de teste. Os casos de testes são construídos de acordo com as especificações das funcionalidades, e levam em consideração os *inputs* e os *outputs*. Cada caso de teste precisa ter uma definição clara do objetivo de teste (LEWIS, 2000). Ainda segundo Lewis (2000), existem os testes de caixa-preta, ou testes funcionais, em que as condições de testes são baseadas nas funcionalidades do sistema, em que o usuário insere os *inputs* e observa os *outputs*, mas sem saber como o programa ou sistema funciona internamente. O testador se concentra em testar a funcionalidade do sistema em relação ao que foi especificado, mas vendo o programa como uma "caixa-preta", despreocupado com sua estrutura interna. Existem também os testes de caixa-branca, ou testes estruturais, que são desenhados para examinar partes lógicas internas do sistema. O testador conhece a estrutura e a lógica interna do programa e especifica um conjunto de entradas e analisa suas saídas. O foco do teste de caixa-branca é o código.

Os testes ainda podem ser manuais, testados por pessoas, ou automatizados, quando executados automaticamente por computadores. Os testes também podem ser dinâmicos ou estáticos. Os estáticos fazem inspeções sem haver necessariamente execução manual ou automatizada do produto que está sendo testado. Alguns exemplos são: análises de sintaxe de código, inconformidade de padrões. Por sua vez, os testes dinâmicos são dependentes da execução de ao menos uma seqüência específica de código.

2.5 Medição e Análise

2.5.1 Conceitos básicos

O processo de medição tem evoluído no contexto de Engenharia de Software. Muitas organizações tratavam medições como um trabalho adicional, no entanto, há uma tendência em organizações bem sucedidas em implementá-la como uma disciplina próativa (McGarry, 2002 apud HAZAN; LEITE, 2003).

A importância da medição começou a ser identificada e estudada há muito algum atrás. DeMarco (1986) afirma que "Não se pode controlar aquilo que não pode medir". Pressman (2006) afirma que "um elemento chave de qualquer processo de engenharia é a medição". Ainda segundo o autor, medidas de software não são como nas demais engenharias as quais estão associadas a leis quantitativas básicas da física, como massa, voltagem, temperatura ou velocidade. Em software essas medidas são freqüentemente indiretas, abertas a debate. Segundo Fenton e Pfleeger (1997) "Medição está se tornando uma parte importante do gerenciamento de projetos de software". As medições de software auxiliam gestores de projeto a identificar oportunidades de melhoria de processo, e ajudam a quantificar características do software (PAVUR; JAYAKUMAR; CLAYTON, 2000) e podem auxiliar na tomada de decisões gerenciais (STARK; DURST; VOWELL, 1994; McGarry, 2002).

Sobre a definição de medição, Fenton e Pfleeger (1997) diz que "Medição é o processo pelo qual números ou símbolos são associados aos atributos de entidade do mundo real de modo que os determinem de acordo com regras claramente definidas". A Entidade se trata, no caso, do objeto a ser caracterizado pela medição de seus atributos. Atributo se caracteriza por ser uma propriedade distinguível de uma entidade. O autor ainda afirma que tentar "medir o não mensurável" a fim de nos ajudar a entender entidades particulares é tão potente em engenharia de software como em qualquer outra disciplina.

Os termos medida, medição, métrica são freqüentemente confundidos e utilizados intercambiadamente. Contudo, é importante entender as diferenças entre tais termos (PRESSMAN, 2006). Segundo o autor a palavra medida pode ser utilizada como substantivo e como verbo. Tratando-se do contexto de engenharia de software medida fornece uma indicação quantitativa da extensão, quantidade, dimensão, capacidade ou tamanho de algum atributo de um processo ou de um produto. Segundo a ISO/IEC 15939 (2007):

- Medição (substantivo): Conjunto de operações que têm o objetivo de determinar o valor de uma medida
- Medição (verbo): Realizar uma medição.
- Medida: Variável a qual é atribuído um valor como o resultado de uma medição.

- Medida básica: Medida definida em termos de um atributo e seu método de quantificação.
- Medida derivada: Medida definida como uma função de duas ou mais medidas básicas.
- Indicador: Medida que fornece uma estimativa ou avaliação de atributos referidos derivados a partir de um modelo com respeito às necessidades de informação definidos.

Takara, Bettin e Toledo (2007) complementam que indicador é uma medida extremamente importante para a organização, sendo analisada e acompanhada periodicamente a partir de uma meta de desempenho previamente estabelecida.

2.5.2 GQM - Goal Question Metric

É comum encontrar atividades de medição que medem apenas o que é conveniente ou fácil de medir. Entretanto, tais medições nem sempre representam o que é realmente necessário a medir. Portanto muito desses programas de medição falham, pois os dados coletados não são úteis por não se saber o que fazer com tais dados (FENTON; PFLEEGER, 1997). Segundo Monteiro (2008) "estudos apontam exemplos de fracassos na implantação de programas de medições nas organizações, devido à definição de um grande número de medidas inúteis, muito complexas, não padronizadas e desalinhadas com a estratégia da organização". O autor conclui que é importante, então, a identificação de medidas relevantes para a avaliação de desempenho dos processos envolvidos no desenvolvimento de software. Fenton e Pfleeger (1997) afirmam que programas de medição podem obter mais sucesso se forem desenvolvidos guiados a objetivos. Existem diferentes abordagens de medição e a GQM é uma das mais comuns. A abordagem GQM fornece essa orientação a objetivos. Esse paradigma é baseado na idéia que a medição deve ser orientada a objetivos para que toda coleta de dados seja baseada num fundamento lógico, fazendo que, assim, medidas relevantes úteis sejam levantadas a fim de suportar os interesses de necessidade de informação do processo de medição (BASILI; CALDIERA; ROMBACH, 1994b). A abordagem GQM foi desenvolvida inicialmente por Basili e Rombach juntamente com o Software Engineering Laboratory (SEL) da agência espacial NASA (BASILI et al., 2002) e utilizado com sucesso em algumas empresas como, por exemplo, a Motorola (DASKALANTONAKIS, 1992).

A abordagem ou paradigma GQM tem três níveis (BASILI; CALDIERA; ROMBACH, 1994a) numa abordagem top-down de definição e bottom-up na interpretação:

1. Nível Conceitual: Os objetivos são definidos de acordo com o contexto em que se aplicará a medição.

2. Nível Operacional: Baseado nos objetivos anteriormente definidos, um conjunto de questões é definido para caracterizar o que se deve responder para que se satisfaçam as necessidades de informação de tais objetivos de medição definidos anteriormente.

3. Nível Quantitativo: Um conjunto de medidas é associado com cada questão definida a fim de respondê-las quantitativamente.

É possível observar essa arquitetura hierárquica na Fig. (1):

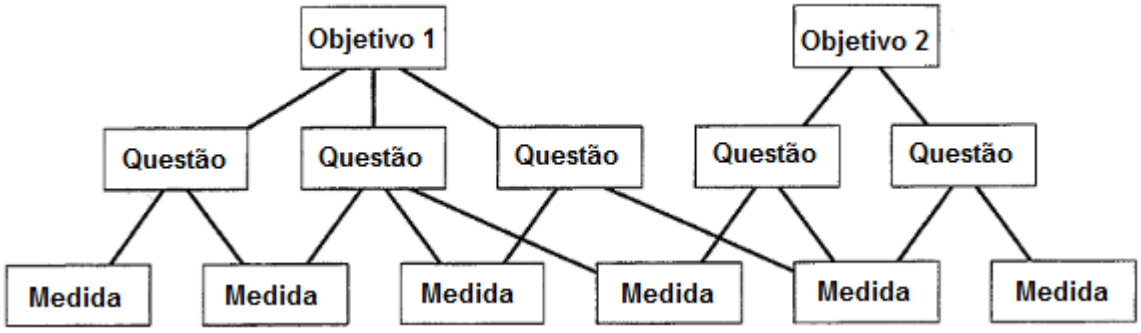

Figura 1 – Arquitetura Hierárquica GQM traduzida (BASILI; CALDIERA; ROMBACH, 1994a)

Observa-se também uma relação com alguns critérios propostos pela ISO/IEC 15939 (2007) em que as medidas devem ter relevância para necessidades de informação levantadas e priorizadas. A ISO/IEC 15939 (2007) também levanta alguns outros critérios a serem considerados ao identificar medidas, tais como:

- Viabilidade de coletar os dados na unidade organizacional
- Disponibilidade de ferramentas adequadas
- Facilidade de interpretação pelo usuário
- Sensibilidade ao contexto
- Grau de intrusão e interrupção de atividades da equipe

A Fig. (2) mostra um exemplo de objetivos, questões e medidas levantadas com a abordagem GQM (BASILI; CALDIERA; ROMBACH, 1994a).

2.5.3 Medidas de Software

São inúmeras as medidas de software já propostas. Um trabalho realizado por Monteiro (2008) fez um levantamento de medidas relacionadas a processo e produto de

Objetivo	Propósito	Melhorar
	Problema	A pontualidade do
	Objeto (processo)	Processo de solicitação de mudança
	Ponto de vista	Do ponto de vista do Gerente
Questão	Q1	Qual é a velocidade de processamento da solicitação de mudança?
Medidas	M1	Média do tempo do ciclo
	M2	Desvio Padrão
	M3	% de casos acima do limite superior
Questão	Q2	O processo de solicitação de mudança (documentação) é realmente realizado?
Medidas	M4	Classificação subjetiva do Gerente de projeto
	M5	% das exceções identificadas durante a revisão
Questão	Q3	Qual é o desvio padrão do tempo real do processo de solicitação de mudança para o estimado?
Medidas	M6	$\frac{Tempo\ médio\ do\ ciclo\ atual - Tempo\ médio\ do\ ciclo\ estimado}{Tempo\ médio\ do\ ciclo\ atual} * 100$
	M7	Avaliação Subjetiva do Gerente de projeto
Questão	Q4	A performance do processo está melhorando?
Medidas	M8	$\frac{Tempo\ médio\ do\ ciclo\ atual}{Tempo\ médio\ de\ ciclo\ da\ baseline} * 100$
Questão	Q5	A performance atual é satisfatória do ponto de vista do Gerente de Projeto?
Medidas	M7	Avaliação Subjetiva do Gerente de projeto
Questão	Q6	A performance está melhorando visivelmente?
Medidas	M8	$\frac{Tempo\ médio\ do\ ciclo\ atual}{Tempo\ médio\ de\ ciclo\ da\ baseline} * 100$

Figura 2 – Exemplo aplicação GQM (BASILI; CALDIERA; ROMBACH, 1994a)

Categoria	Quantidade de medidas
Qualidade	314
Escopo	252
Tempo	109
Esforço	87
Custo	82
Produtividade	22
Não categorizadas	3

Tabela 1 – Medidas por categorias encontradas por Monteiro (2008)

software, envolvendo categorias como qualidade, escopo, tempo, esforço, custo, produtividade, e foi identificado um total de 869 medidas, sendo categorizadas de acordo com a Tab. (1) (MONTEIRO, 2008).

Monteiro (2008) e Trodo (2009) fazem um levantamento de várias medidas de software em seus trabalhos. Hutcheson (2003) em seu livro faz um levantamento algumas medidas. Na Fig. (3) é possível observar alguns exemplos de medidas de software.

Na Fig. (3) Grady (1994) é possível observar medidas de software utilizadas em

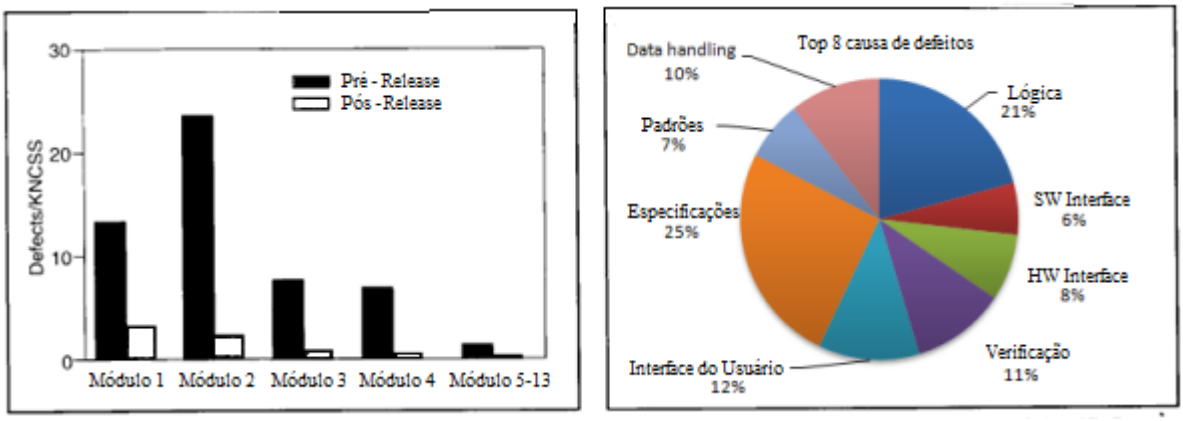

Figura 3 – Exemplo defeitos por módulo e causas mais comuns - traduzida e adaptada (GRADY, 1994)

duas empresas. Ao lado esquerdo da figura tem-se uma divisão por módulos do software em que são explicitados, de um projeto, a quantidade de defeitos identificados antes e depois da entrega do produto. Ao lado direito da figura existe um gráfico com as oito maiores causas de defeitos nos projetos da empresa.

Na Fig. (4) é possível observar um indicador de defeitos por tipo Hutcheson (2003) e citado no trabalho de Trodo (2009).

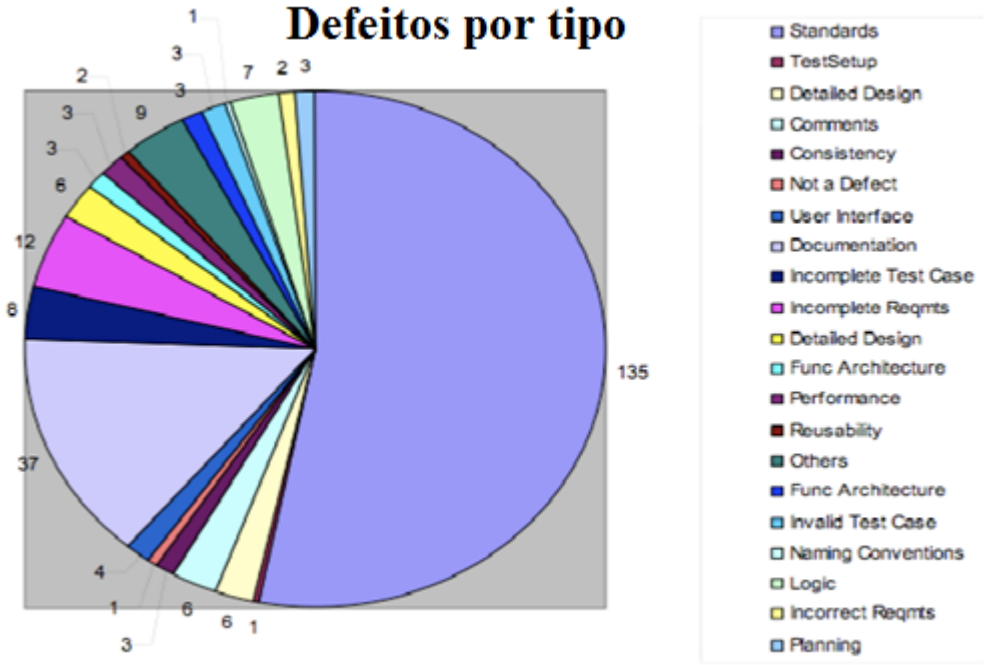

Figura 4 – Exemplo defeitos por tipo. (HUTCHESON, 2003)

Na Fig. (5) observa-se um exemplo de classificações de severidade de defeitos definido por tipo Hutcheson (2003).

Baseado na classificação de defeitos definidas na Fig. (5) é possível ter um indicador de quantidade de defeitos conforme unidades do sistema e logo após de distribuição de

NÍVEIS DE SEVERIDADE	CRITÉRIOS POR SEVERIDADE
Severidade 1	Programa cessa em uma operação significativa
Severidade 2	Erro grave de funcionalidade mas a aplicação pode continuar
Severidade 3	Resultado inesperado ou operação inconsistente
Severidade 4	Visual Design ou sugestão

Figura 5 – Exemplo classificação de defeitos por severidade - traduzida (HUTCHESON, 2003)

defeitos por severidade Hutcheson (2003) conforme pode-se observar nas Fig. (6) e (7).

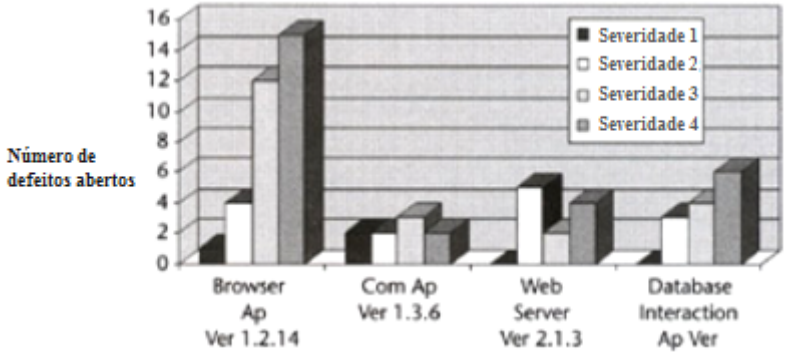

Figura 6 – Exemplo distribuição de severidade de defeitos por módulos (HUTCHESON, 2003)

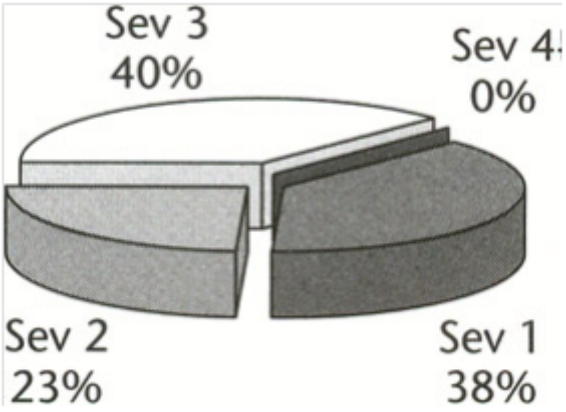

Figura 7 – Exemplo distribuição de defeitos por severidade (HUTCHESON, 2003)

2.6 Considerações Finais

A partir da revisão bibliográfica foi possível observar a importância da Qualidade de Software no contexto de gestão de projetos, bem como a importancia das atividades relacionadas à Gestão da Qualidade, como as de Teste de Software. Também foram evidenciadas dificuldades que os gestores de projetos encontram ao desempenhar suas atividades de gerência, principalmente por software ser um produto abstrato. Apesar dessa

característica incomum, observa-se que medidas podem auxiliar estes gestores a quantificar atributos do projeto de software, fazendo com que tenham mais ciência do estado do produto e possam também identificar oportunidades de melhoria no processo de desenvolvimento, além de servirem como auxilio ao processo de tomada de decisões gerenciais.

3 Metodologia

Tendo em vista o contexto apresentado anteriormente, o qual este trabalho está inserido, e a revisão bibliográfica, evidenciou-se a importância da medição em gestão de projetos. Sendo assim, a proposta deste trabalho é o levantamento de medidas e indicadores que possam auxiliar os gestores de projetos a terem uma visão mais clara, simples e objetiva do estado da qualidade do software, de acordo com os testes realizados, ao longo do ciclo de vida de desenvolvimento do software. E, utilizando as medidas levantadas, se possa, num segundo momento, desenvolver uma aplicação que colete as medidas definidas e as apresente num painel de acompanhamento para os interessados. Assim, foi definido o fluxo de atividades conforme a Fig (8) e Tab. (2).

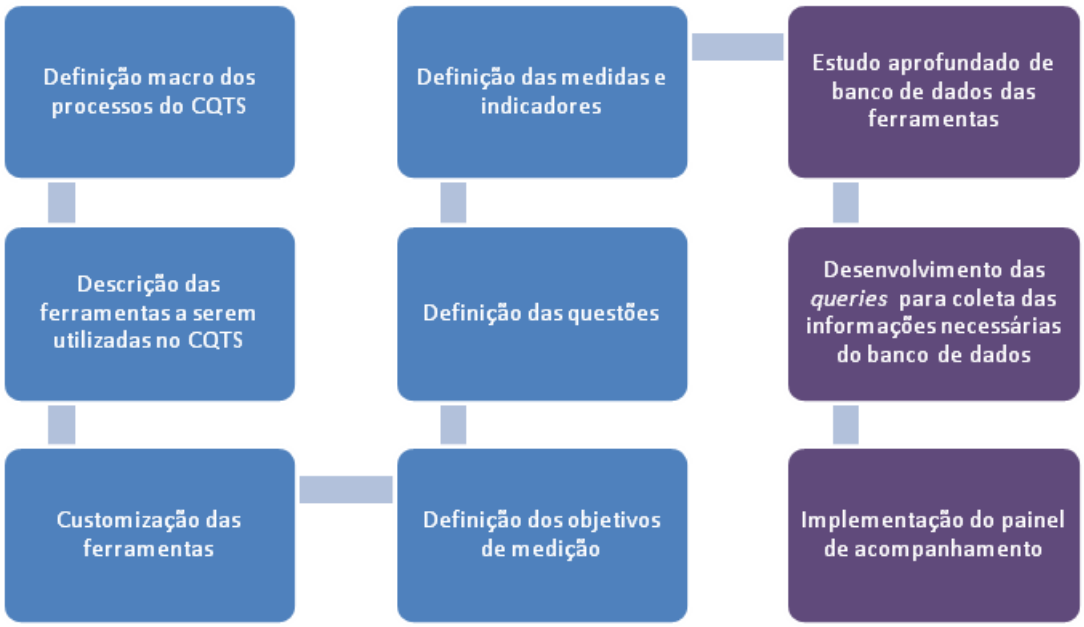

Figura 8 – Fluxo de atividades a serem desenvolvidas ao longo deste trabalho

Para o levantamento das medidas e indicadores será utilizada uma abordagem baseada no paradigma GQM, dada também experiência prévia nessa abordagem, em que as medidas não serão definidas isoladamente, conforme citado na Seção 2.5, mas serão derivadas das atividades de Definição dos objetivos de medição, Definição das questões e Definição das medidas e indicadores, realizando-se assim uma rastreabilidade entre objetivos, questões e medidas.

Passo	Explicação
Definição macro dos processos CQTS	Definição macro dos processos do CQTS relacionados a atividades de testes para entender quais as necessidades da empresa no que tange seu próprio processo de testes.
Descrição das ferramentas a serem utilizadas no CQTS	Estudo e descrição das ferramentas de gestão de testes e defeitos a serem implantadas no CQTS.
Customização das ferramentas	Tendo em vista as necessidades observadas na definição do processo será realizada a customização das ferramentas de gestão de testes e defeitos.
Definição dos objetivos de medição	Levantamento dos objetivos de medição segundo abordagem GQM.
Definição das questões	Levantamento das questões relacionadas aos objetivos de medição.
Definição das medidas e indicadores	Levantamento das medidas e posteriormente indicadores, relacionados às questões definidas anteriormente.
Estudo aprofundado de banco de dados das ferramentas	Estudo das tabelas e suas relações do banco de dados das ferramentas de gestão de testes e defeitos, a fim de viabilizar a coleta de dados para as medidas e indicadores a serem implantadas no painel de acompanhamento.
Desenvolvimento das queries para coleta das informações necessárias do banco de dados	Desenvolvimento das queries em linguagem SQL para a pesquisa das informações necessárias para a construção dos indicadores
Implementação do painel de acompanhamento	Implementação da ferramenta que coletará, através do banco de dados, as medidas, e apresentará os indicadores levantados para o usuário.

Tabela 2 – Descrição do Fluxo de atividades a serem desenvolvidas ao longo deste trabalho

4 O painel de acompanhamento de projetos: Proposta

Este Capítulo destina-se à apresentação da primeira etapa do trabalho, no que diz respeito às seis primeiras atividades definidas na Tab. (2) e Fig. (8) , conforme a metodologia apresentada no Capítulo 3. Na Seção 4.1 serão apresentadas as definições macros dos dois principais processos que estão atualmente sendo executados no CQTS: o de recebimento de demanda e o processo de teste. A realização dessa definição tem como objetivo gerar o entendimento do funcionamento dos processos do CQTS a fim de gerar insumos para a implantação de ferramentas e medidas adequadas. Na Seção 4.2 serão apresentadas as ferramentas de gestao de testes e defeitos que serão implantadas no laboratório para que os processos descritos anteriormente possam ser auxiliados por tais ferramentas. Finalmente na Seção 4.3 serão apresentadas as medidas definidas, baseadas através da abordagem GQM, que poderão ser disponibilizadas através de insumos coletados das ferramentas descritas, e os indicadores gerados através das medidas definidas.

4.1 Processos do CQTS

O Centro de Qualidade e Testes de Software (CQTS) é um laboratório recém criado da Faculdade Gama (FGA) da Universidade de Brasília (UnB) no contexto do curso de graduação em Engenharia de Software. Um dos objetivos do laboratório é realizar pesquisa e desenvolvimento em tecnologias que buscam prover uma infraestrutura de qualidade de software adequada que apoie o processo de produção e manutenção de sistemas de software.

O laboratório está, atualmente, trabalhando em parceria com uma grande empresa nacional de desenvolvimento de produtos de *hardware* e *software*, mais especificamente relacionados com dispositivos móveis, como *smartphones* e *tablets*, em suas diversas plataformas, sistemas operacionais e aplicativos.

Algumas das frentes de trabalho do CQTS são:

- Execução de casos de teste
- Construção de plano de testes e casos de testes
- Automatização de testes

- Gestão de testes

Dentre alguns benefícios do CQTS, pode-se citar que o laboratório possibilita a integração e interação entre os estudantes integrantes do laboratório com profissionais do mercado de trabalho, com envolvimento em projetos reais, fazendo com que haja equilíbrio entre aprendizado, pesquisa acadêmica, e realização de atividades reais do mercado de trabalho, preparando assim melhor os estudantes para a vida profissional. No contexto deste trabalho, o CQTS proverá um ambiente que permita que seja feita a aplicação em um contexto onde há interação entre indivíduos, processos e ferramentas, fazendo com que a proposta possa ser aplicada em um ambiente real.

O laboratório recebe demandas, sendo a maioria delas relacionadas à execução, à construção ou à automatização de testes. Algumas demandas podem possuir prioridade maior que as outras e, este fator, juntamente com os fatores de complexidade, tamanho e prazo da demanda, faz com que seja necessária fazer uma escolha adequada da equipe que será alocada para o desempenho de tais demandas. Na Figura (9) é apresentado o processo de gestão de demandas do CQTS.

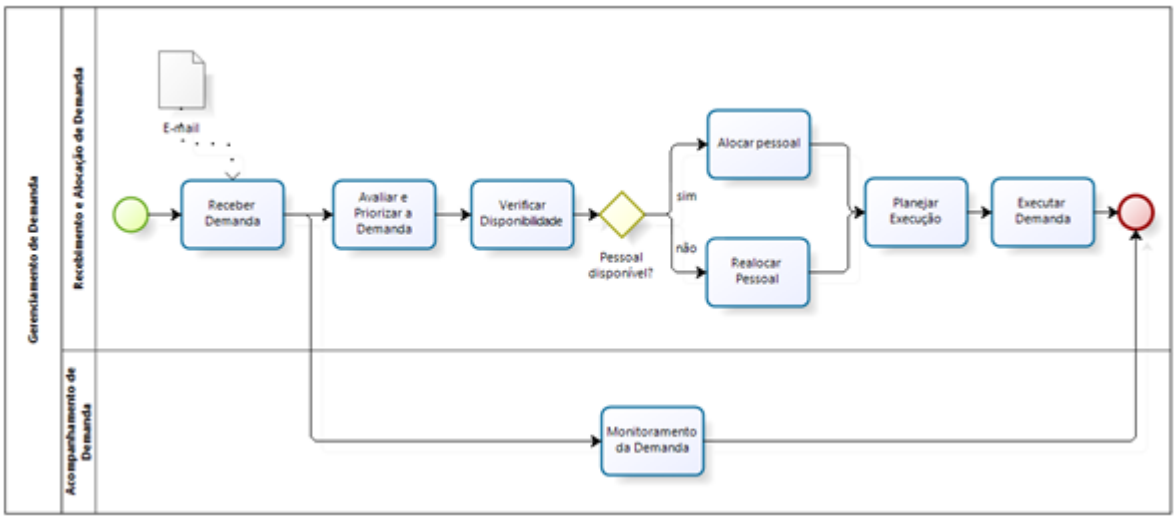

Figura 9 – Processo de gestão de demandas do CQTS

A atividade de planejamento da execução consiste em dividir a demanda em tarefas menores para que se possa ter uma melhor visibilidade sobre as atividades que devem ser executadas. Uma demanda de teste costuma ser de execução, construção ou customização de casos de teste. Na Figura (10) observa-se como se dá o processo de uma demanda de execução de testes.

As demandas de construção de casos de teste seguem o fluxo demonstrado na Fig. (11).

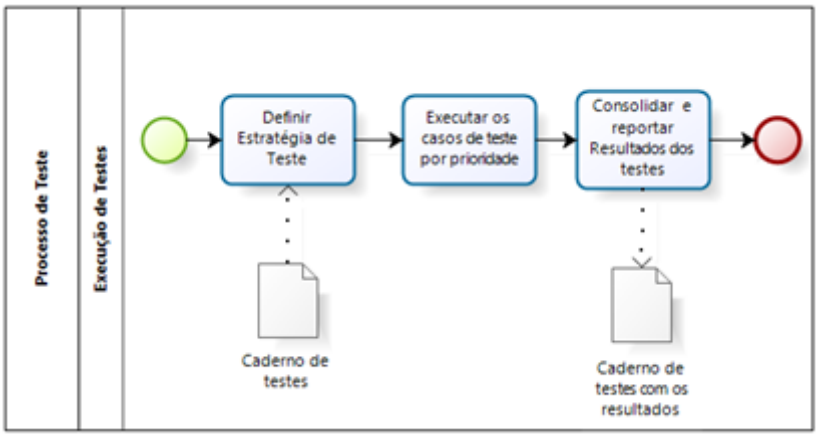

Figura 10 – Processo de execução de testes do CQTS

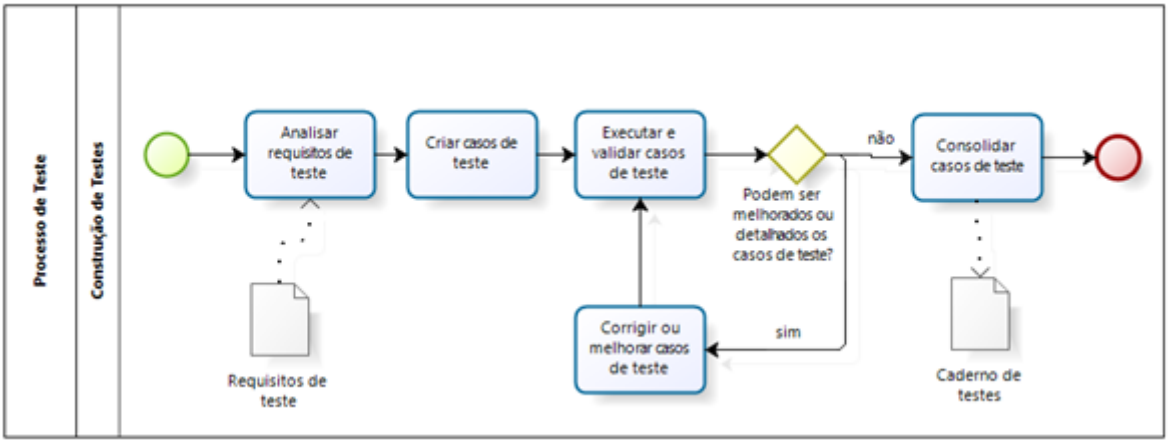

Figura 11 – Processo de construção de testes do CQTS

Já as demandas de automatização de casos de teste seguem o seguinte fluxo conforme a Fig. (12).

4.2 Ferramentas

4.2.1 Descrição de Ferramentas a serem utilizadas no CQTS

O controle dos fluxos de trabalho está sendo realizado com uma ferramenta de gestão, chamada Redmine que é uma ferramenta web e open source. Alguns recursos disponibilizados pela ferramenta são o calendário e os gráficos de *Gantt*, que são utilizados para demonstrar como está o andamento das demandas e suas atividades. As demandas são divididas como projetos internos ou projetos externos e têm as classificações citadas anteriormente, como execução, construção ou automatização de testes. De acordo com a necessidade e surgimento de demandas de caráter distinto, outras classificações podem ser geradas. Na atividade de planejar a execução da demanda, citado anteriormente na

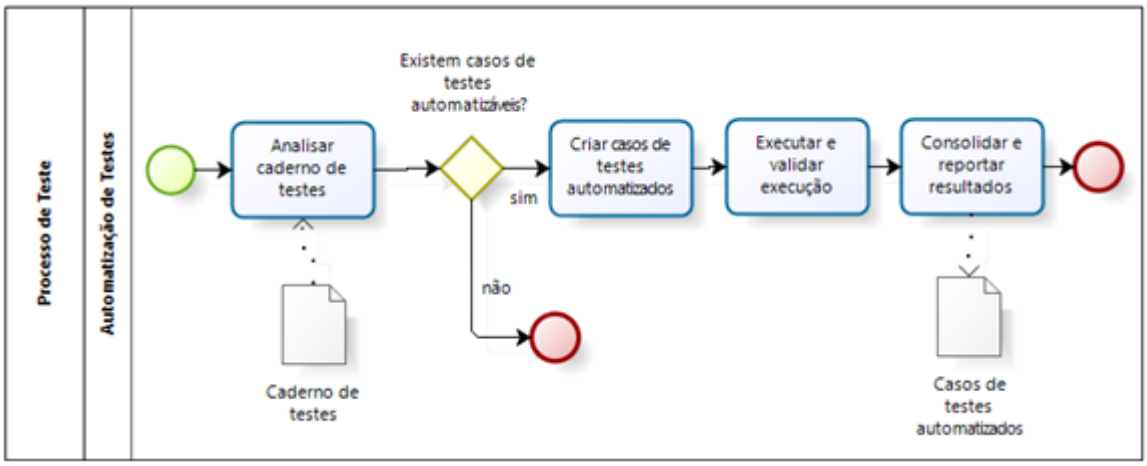

Figura 12 – Processo de automatização de testes do CQTS

descrição do processo, a demanda é dividida em tarefas e cada tarefa é atribuída a um responsável. A ferramenta permite realizar melhor o controle de alocação de pessoas, além de gerar relatórios de atividades, que podem ser adicionados conforme o usuário atualiza o estado atual de suas tarefas.

Devido o alto volume de demandas de execução de testes que o CQTS recebe, percebeu-se a necessidade do uso de ferramentas que apóiem o processo de execução e gestão de testes. Foi realizado, como um projeto interno do CQTS, um estudo comparativo entre ferramentas open source de gestão de testes, a fim de se identificar a ferramenta ou o conjunto de ferramentas que pudessem dar suporte à equipe de testadores e aos gestores dos projetos. A seleção inicial das ferramentas foi baseada, dentre outros, nos seguintes requisitos:

- Criação de casos de testes;
- Criação de plano de testes
- Execução de casos de testes;
- Gestão de defeitos;

As ferramentas selecionadas para a avaliação da usabilidade e da qualidade em uso foram os pares Testlink[1] com Mantis[2] e Bugzilla[3] com Testopia[4]. O Bugzilla é um sistema de rastreamento de defeitos, em que são cadastrados produtos e defeitos vinculados a esses produtos, sendo possível, assim, mudar o estado dos defeitos, anexar informações, responsáveis, dentre outras funcionalidades. O Testopia é um plugin para o Bugzilla em

[1] Testlink: http://teamst.org/
[2] Mantis Bug Tracker: http://www.mantisbt.org/
[3] Bugzilla: http://www.bugzilla.org/
[4] Testopia: https://developer.mozilla.org/en-US/docs/Mozilla/Bugzilla/Testopia

que planos de testes e casos de testes são vinculados a esses produtos, podendo realizar um mapeamento entre os testes e os defeitos gerados pela execução desses testes. Os critérios para a avaliação das ferramentas foram selecionados considerando-se critérios estabelecidos na norma ISO/IEC 9126, por meio de medidas de software referentes à Qualidade em Uso e à característica de qualidade "Usabilidade", por se considerar que estas possuem impacto no contexto do CQTS. Após este estudo o par Bugzilla e Testopia foi considerado mais adequado para apoiar o processo de teste do CQTS.

4.2.2 Customização de Ferramentas utilizadas no CQTS

O laboratório CQTS mantém uma parceria com uma grande empresa nacional do ramo de desenvolvimento de software e hardware, sendo ela a principal fonte de demandas de execução de testes do laboratório. Esta empresa tem definido o processo interno de execução de testes e gestão de defeitos, mas utiliza apenas ferramenta para gestão de defeitos, não utilizando uma ferramenta específica para gestão de testes. Assim, a gestão e execução desses testes é feita através de planilhas.

Uma das atividades que estão sendo realizadas pelo CQTS é a análise da ferramenta de gestão de defeitos utilizada pela empresa e um mapeamento do seu processo interno, visando aferir a aderência do processo atual à ferramenta de gestão de defeitos Bugzilla.

A ferramenta Bugzilla disponibiliza um conjunto de campos padrão que não descrevem completamente o defeito de acordo com as necessidades levantadas, conforme observa-se na Fig. (13). Pode-se observar na Fig. (14) o conjunto customizado de campos ao se cadastrar um novo defeito.

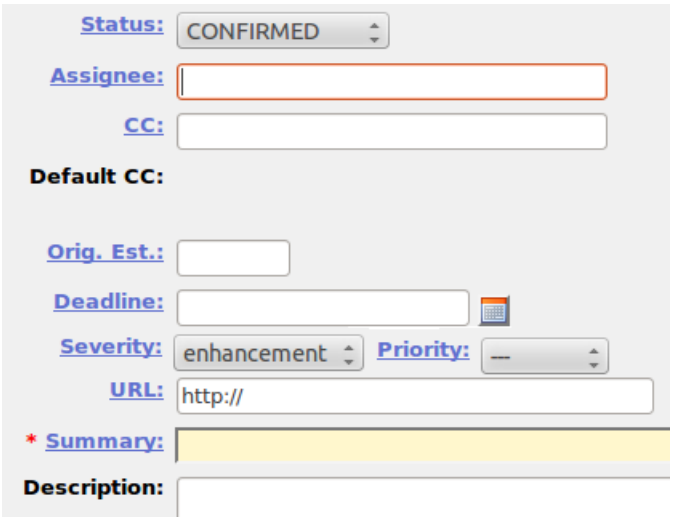

Figura 13 – Exemplo de campos padrões do Bugzilla

A ferramenta Bugzilla conta com muitas opções de customização em seu painel administrativo, fazendo com que os campos que se mostravam diferentes entre as ferramentas

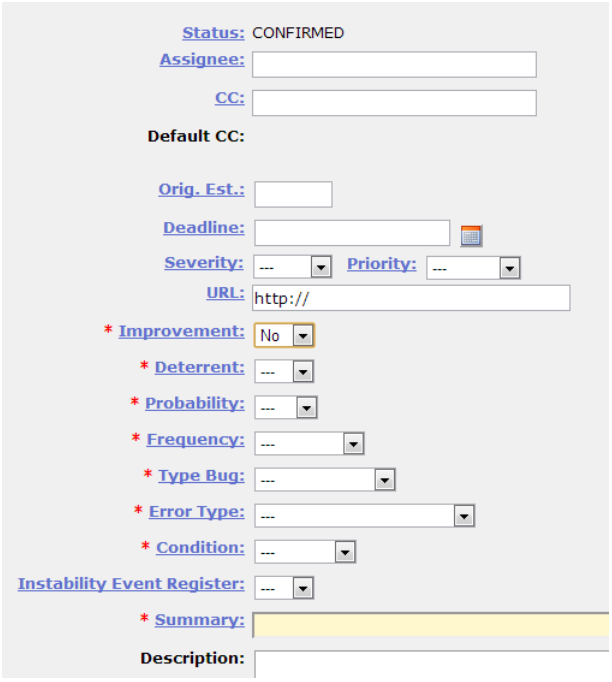

Figura 14 – Campos do Bugzilla customizados

pudessem ser adaptados, e os campos ausentes criados. O mapeamento e a adaptação dos campos na ferramenta de gestão de testes não são tão simples devido as poucas opções de customização disponibilizadas pelo Testopia. Portanto, é necessário um trabalho de adaptação e aproveitamento das informações disponibilizadas pela ferramenta. Pode-se observar na Fig. (15) os campos disponíveis na criação de um novo caso de teste.

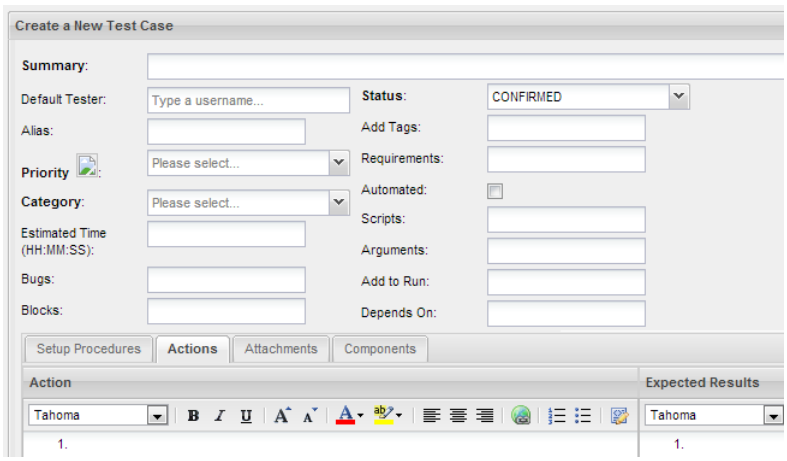

Figura 15 – Campos ao criar caso de teste do Testopia

4.3 Levantamento das Medidas

4.3.1 Objetivos de Medição

Levando em consideração o contexto do CQTS e alinhados a seus objetivos relacionados à qualidade de software e melhoria de processos, foram levantados os objetivos de medição mostrados nas Tabs. (3) e (4).

O1. Objetivo I	
Analisar:	O produto de software e os defeitos encontrados pelas atividades de Testes de Software
Para o propósito de:	Entender
Com respeito a:	Evolução da qualidade
Do ponto de vista do:	Gestor do projeto
No contexto de:	Desenvolvimento e/ou manutenção de software

Tabela 3 – Objetivo de medição I

O2. Objetivo II	
Analisar:	O produto de software e os defeitos encontrados pelas atividades de Testes de Software
Para o propósito de:	Entender
Com respeito a:	Rastreabilidade
Do ponto de vista do:	Gestor do projeto
No contexto de:	Desenvolvimento e/ou manutenção de software

Tabela 4 – Objetivo de medição II

Um visão da evolução da qualidade do produto é importante aos gestores para que sejam providas informações a respeito do quanto do produto já foi testado e quantos/quais defeitos foram corrigidos. Tal visibilidade pode servir como base para auxiliar os gestores a defenir se uma release oficial do produto pode, ou não, ser entregue ao cliente, bem como pode auxiliá-los a identificar possíveis desvios ou baixas na evolução da qualidade. A obtenção de uma visão de rastreabilidade dos tipos de defeitos encontrados é importante para que tais gestores possam ter insumos para identificar os defeitos e suas possíveis origens. Ao identificar tais origens, pode-se tomar ações de mitigação das causas raízes a fim de minimizar ou até eliminar a ocorrência de novos defeitos.

4.3.2 Questões

Tendo em vista os objetivos descritos anteriormente, foram definidas algumas questões, conforme demonstrado na Tab. (5).

Pode-se observar nas Tabs. (6) e (7) a ligação entre os objetivos previamente descritos e as questões definidas.

ID da Questão	Questão
Q1	Que tipos de defeitos são mais encontrados?
Q2	Qual gravidade dos defeitos encontrados?
Q3	Os defeitos estão sendo tratados e corrigidos?
Q4	O produto foi testado suficientemente?
Q5	Como está a evolução da qualidade do software ao longo do tempo?

Tabela 5 – Definição das questões

Objetivo	Questões
O1. Evolução da Qualidade	Q2. Qual gravidade dos defeitos encontrados? Q3. Os defeitos estão sendo tratados e corrigidos? Q4. O produto foi testado suficientemente? Q5. Como está a evolução da qualidade do software ao longo do tempo?

Tabela 6 – Rastreabilidade entre questões e objetivo I

Objetivo	Questões
O2. Rastreabilidade	Q1. Que tipos de defeitos são mais encontrados?

Tabela 7 – Rastreabilidade entre questões e objetivo II

4.3.3 Medidas

4.3.3.1 Medidas Básicas

Para que as questões definidas na Seção 4.3.2 possam ser respondidas, foram definidas inicialmente algumas medidas básicas, conforme descritas na Tab. (8).

Além disso, pode-se definir diretrizes para interpretação dessas medidas básicas:

1. Quantidade de Defeito: Um número grande de defeitos pode indicar software de baixa qualidade, requisitos mal definidos ou também testes mal escritos. Entretanto essa medida isoladamente pode não fornecer informação muito relevante tendo em vista que um software X pode conter muitos defeitos e um software B poucos defeitos, entretanto os defeitos do software X, apesar de muitos, podem ser apenas defeitos cosméticos enquanto os defeitos do software Y, apesar de poucos, podem ser defeitos mais graves. Há também a possibilidade do software X, apesar de ter o dobro de defeitos, ter também 60 vezes o tamanho do software Y, fazendo com que, apesar da quantidade maior de defeitos, tenha na verdade, proporcionalmente, menos defeitos.

2. Severidade de defeito: Medida básica de um defeito. Um defeito grave significa um defeito com impacto maior no funcionamento do software em relação a um defeito médio ou leve. Essa medida usada em conjunto com outras medidas, como quanti-

ID	Medida	Descrição
MB1	Quantidade de Defeitos	Quantidade de defeitos encontrados nas atividades de teste.
MB2	Severidade de defeito	Classifica cada defeito de acordo com sua severidade.
MB3	Tipo de Defeito	Classifica o defeito de acordo com o tipo.
MB4	Estado de defeito	Classifica o defeito de acordo com seu estado atual.
MB5	Estado do caso de teste	Classifica o caso de teste de acordo com seu estado atual.
MB6	Tipos de casos de teste	Classifica o caso de teste de acordo com seu tipo.
MB7	Quantidade de casos de teste	Quantidade de casos de teste que existem para um produto.
MB8	Quantidade de ciclos de teste	Quantidade de ciclos de teste executados para um produto.
MB9	Quantidade de linhas de código	Quantidade de linhas de código do software.

Tabela 8 – Medidas básicas definidas

dade de defeitos, por exemplo, pode ser mais relevante na visão geral do estado do produto.

3. Tipo de Defeito: Medida básica de um defeito. Um defeito de lógica pode indicar um erro na implementação ou até mesmo um requisito mal definido ou ambíguo. Essa medida isoladamente não é tão relevante ao contexto.

4. Estado de defeito: Medida básica de um defeito. Um defeito fechado indica que não se deve, a tempo, preocupar-se com tal defeito. Um defeito aberto indica que o defeito ainda está pendente de solução.

5. Estado do caso de teste: Medida básica de um caso de teste. Um caso de teste no estado 'falha' indica que durante a execução do caso de teste o resultado obtido foi diferente do resultado esperado especificado. Um caso de teste no estado 'aprovado' indica que o resultado esperado foi obtido.

6. Tipos de casos de teste: Um caso de teste pode ser categorizado de acordo com uma característica da qualidade específica como, por exemplo: funcionalidade, interface, estabilidade, segurança.

7. Quantidade de casos de teste: Um número elevado de casos de teste para um produto pode indicar uma boa massa de testes e um número pequeno uma massa de testes pobre para um produto. Contudo, essa medida, isoladamente, não carrega muito valor informativo, entretanto, considerando uma massa de testes de qualidade essa

medida pode auxiliar outras medidas, como por exemplo, a de percentual de casos de testes já executados.

8. Quantidade de ciclos de teste: Medida básica que indica quantas vezes um conjunto de testes de um determinado produto foi rodado.

9. Quantidade de linhas de código: Medida básica de produto de software que indica tamanho do software.

Algumas medidas relacionadas a fase do ciclo de vida que os defeitos foram encontrados não são passíveis de se coletar, pois o CQTS costuma receber os produtos já finalizados ou durante a fase de construção.

4.3.3.2 Medidas Derivadas

As medidas definidas na Seção anterior são medidas básicas e isoladamente não carregam consigo alto valor informativo. Portanto, essas medidas básicas serão utilizadas na definição das medidas derivadas listadas na Tab. (9).

4.3.3.3 Indicadores

Conforme mencionado no Capítulo 2, indicador é uma medida de grande importância para a organização, devendo ser analisada e acompanhada periodicamente. A partir das medidas derivadas definidas na Seção anterior é possível construir alguns indicadores, conforme descrito a seguir:

Total de defeitos encontrados VS Total de defeitos corrigidos

É natural que defeitos sejam encontrados durante as atividades de testes de um software e tais defeitos também devem ser corrigidos. Esse indicador diz respeito à relação entre o total de defeitos já encontrados, num determinado produto, e o total de defeitos já corrigidos deste produto, ao longo do tempo. Colocando esses montantes de defeitos encontrados e corrigidos num gráfico de linhas, as linhas correspondentes a esses dois estados de defeitos devem, idealmente, estar juntas ou o mais próximo possível. Uma simulação próxima ao ideal desse indicador pode ser visto na Fig. (16).

Verifica-se no gráfico da Fig. (16) que conforme os defeitos são encontrados ao longo do desenvolvimento, há também um trabalho de correção desses defeitos que faz com que as duas linhas permaneçam próximas e com o avanço do tempo, vão se aproximando cada vez mais. Isso evidencia que além do trabalho de identificação de defeitos também está havendo um trabalho proporcional de correção de defeitos.

Um cenário diferente do que se espera pode ser observado na Fig. (17).

ID	IDs das medidas básicas utilizadas	Medida	Descrição
MD1	MB1, MB9	Densidade de defeitos	Quantidade de defeitos encontrados em relação ao tamanho do software em linhas de código.
MD2	MB1, MB3	Quantidade de defeitos por tipo	Quantidade de defeitos encontrados em relação ao tipo de defeito.
MD3	MB1, MB2, MB4	Quantidade de defeitos abertos por severidade	Quantidade de defeitos abertos encontrados em relação à severidade.
MD4	MB1, MB2, MB4	Quantidade de defeitos corrigidos por severidade	Quantidade de defeitos corrigidos encontrados em relação à severidade.
MD5	MB1, MB8	Quantidade de defeitos por ciclo de teste	Quantidade de defeitos encontrados no software a cada ciclo de teste.
MD6	MB1, MB4	Taxa de remoção de defeitos	Quantidade de defeitos do software corrigidos em relação ao total de defeitos encontrados.
MD7	MB1, MB4	Quantidade de defeitos reabertos	Quantidade de defeitos do software que foram reabertos
MD9	MB5, MB7	Cobertura de testes	Percentual em que um software foi testado em relação ao número de casos de testes existentes e executados.
MD10	MB5, MB6, MB7	Cobertura de testes por tipo	Percentual em que um software foi testado em relação ao tipo de caso de teste
MD11	MB9	Cobertura de código	Percentual em que um software foi testado a partir da quantidade de linhas de código que são executadas nos testes unitários.

Tabela 9 – Medidas derivadas definidas

Observa-se na Fig. (17) que até aproximadamente o décimo sexto dia as atividades de correção de defeitos estavam com desempenho proporcional à identificação de defeitos. Após esse período, a taxa de correção de defeitos caiu consideravelmente, indicando que houve, ou está havendo, alguma dificuldade em tais atividades. Esse cenário pode indicar, dentre outras coisas, um possível atraso na entrega do produto, pois as atividades de correção de defeitos não estão conseguindo acompanhar o montante de defeitos encontrados.

Esse indicador pode ser incrementado com uma nova linha que contém a quantidade atual de defeitos no estado aberto, que nada mais é que a subtração entre os defeitos encontrados e corrigidos ponto a ponto. Considerando o primeiro caso apresentado, próximo ao ideal, é possível observar na Fig. (18), que conforme o passar do tempo

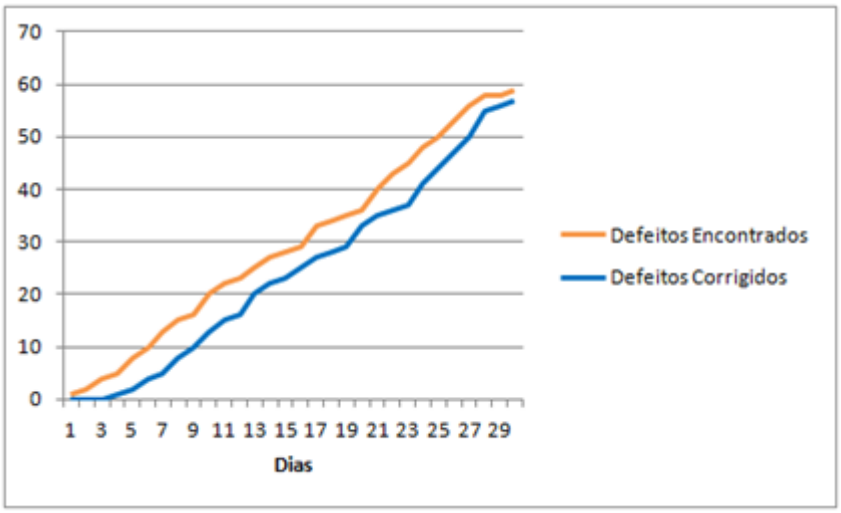

Figura 16 – Defeitos encontrados vs corrigidos - Exemplo 1

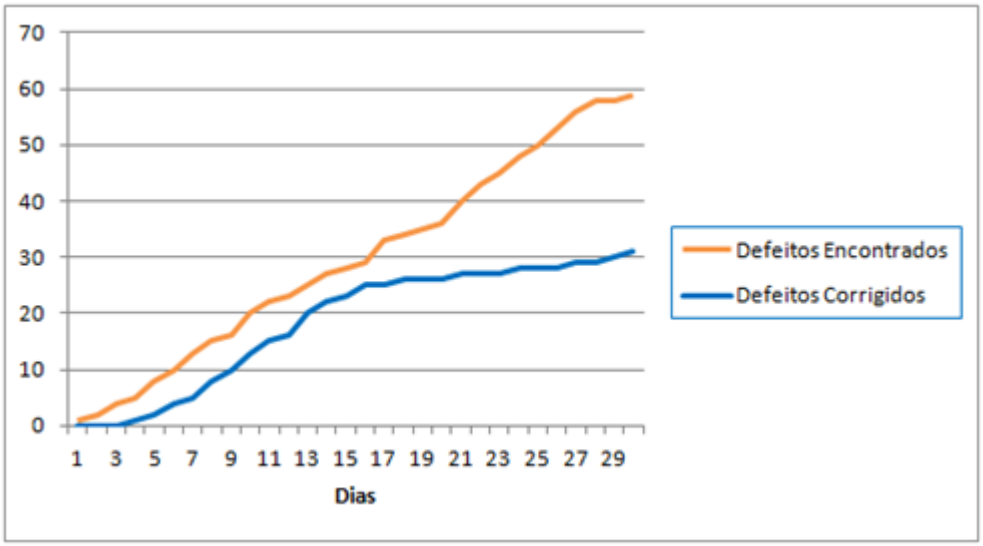

Figura 17 – Defeitos encontrados vs corrigidos - Exemplo 2

a quantidade de defeitos abertos (pendentes de correção) se mantém estável e, no final, se aproxima de zero.

O mesmo gráfico da Fig. (17) também pode ser incrementado com essa nova linha, conforme se observa no gráfico da Fig. (19). Observa-se então, nessa figura, que a partir do décimo sexto dia a quantidade de defeitos abertos começa a subir.

Total de defeitos graves encontrados VS Total de defeitos graves corrigidos

Esse indicador é bem similar ao anterior. Entretanto, ao invés de provar uma visão geral de todos os defeitos encontrados em relação aos corrigidos, ele especializa esses defeitos no tipo de defeito classificado como de severidade grave. Verifica-se um exemplo desse indicador na Fig. (20).

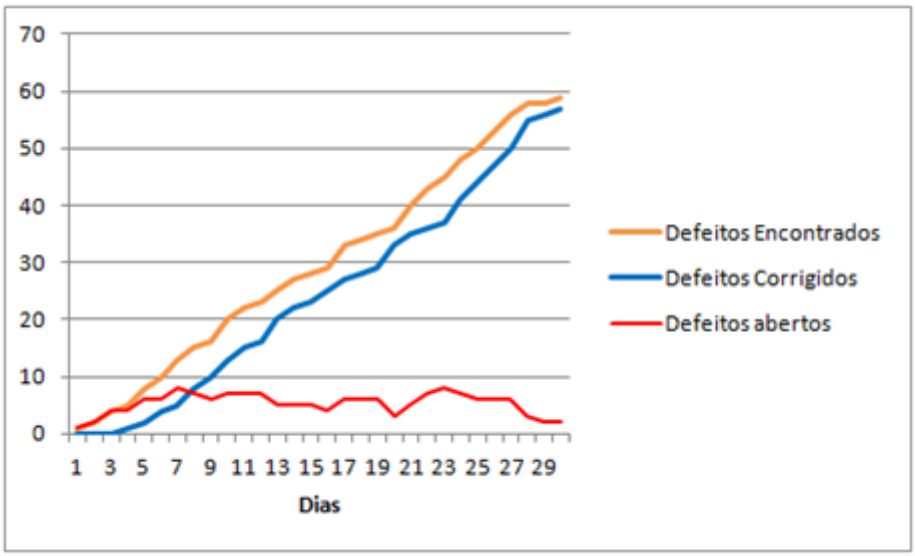

Figura 18 – Defeitos encontrados vs corrigidos (com defeitos abertos) - Exemplo 1

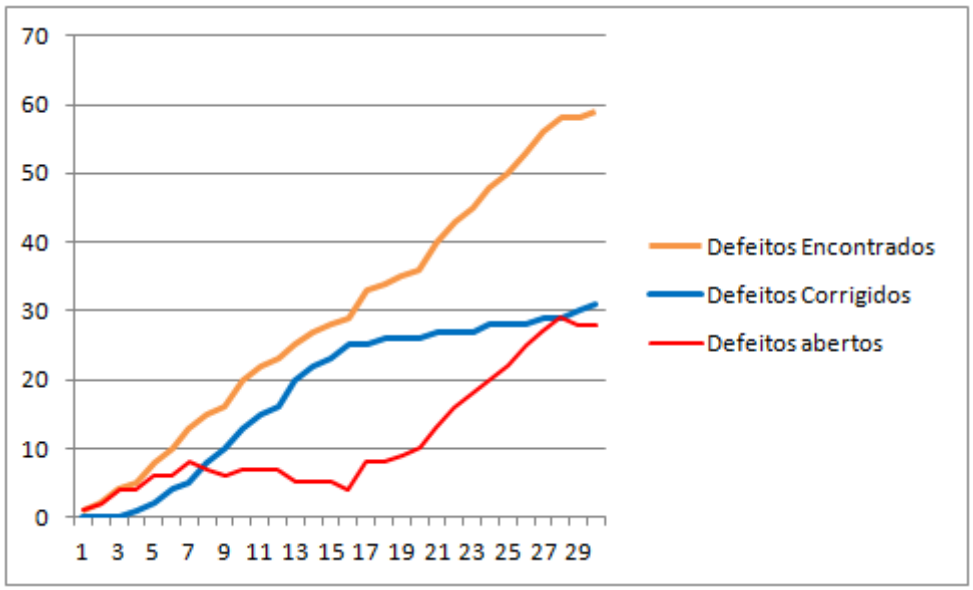

Figura 19 – Defeitos encontrados vs corrigidos (com defeitos abertos) - Exemplo 2

Esse indicador especializado em defeitos graves ajuda a cobrir um uma perspectiva que o indicador passado não cobre completamente, pois, ao considerar apenas a relação entre defeitos já encontrados e defeitos já corrigidos, pode haver o caso em que a maioria dos defeitos corrigidos diz respeito a defeitos cosméticos, pouco relevantes, e que os defeitos realmente graves não vêm sendo corrigidos.

Quantidade de defeitos reabertos de acordo com o tempo

É possível que haja casos em que defeitos, dados como solucionados, voltem a aparecer. Esse indicador mostra a quantidade de defeitos que são reabertos por ciclo de teste. É possível observar um exemplo desse indicador na Fig. (21).

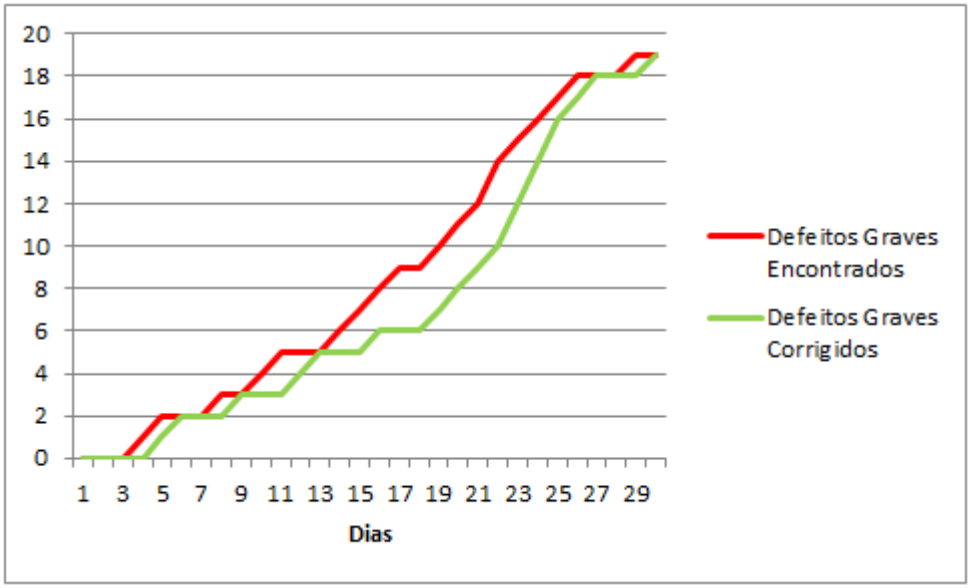

Figura 20 – Defeitos encontrados graves vs graves corrigidos

Figura 21 – Defeitos reabertos por ciclo de testes

Pode-se observar no gráfico da Fig. (21) que nos primeiros ciclos não houve defeitos reabertos. Entretanto, pode-se observar que ao longo dos ciclos de testes, diversos defeitos foram reabertos. Isso pode indicar que os defeitos que anteriormente foram identificados e supostamente corrigidos podem não ter sido realmente corrigidos. Pode indicar também que o código está altamente acoplado e que uma modificação em um módulo do código está afetando outro módulo, fazendo com que erros previamente corrigidos reapareçam. Tendo em mãos essa informação, o gestor do projeto pode verificar esse comportamento e tomar as providências cabíveis.

Densidade de defeitos abertos por severidade

Esse indicador diz respeito à quantidade de defeitos abertos existentes classificados de acordo com a sua severidade. Ele fornece um retrato atual dos defeitos abertos em relação a um projeto. Um exemplo desse indicador está na Fig. (22).

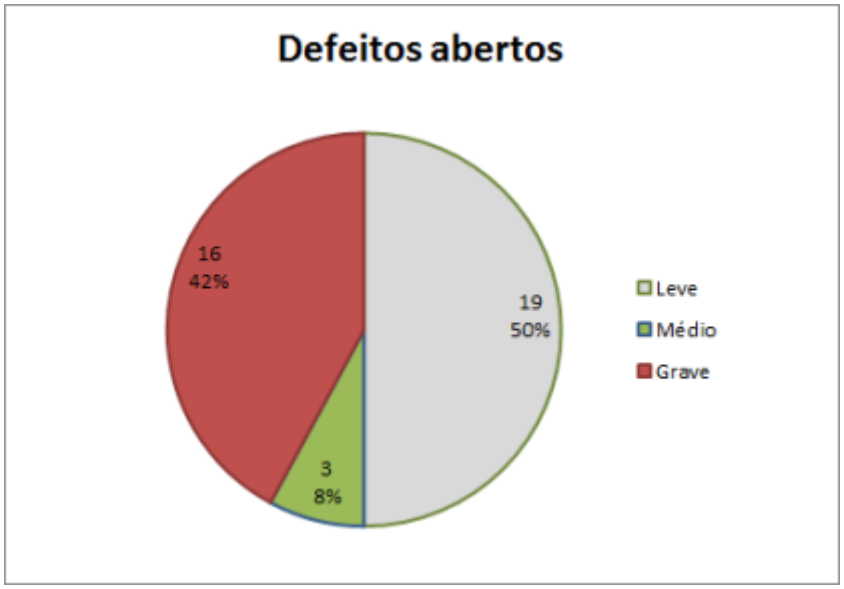

Figura 22 – Defeitos abertos por severidade

Pode-se observar na Fig. (22) que a maior parte dos defeitos é de severidade leve. Entretanto, existe uma parcela considerável de defeitos de severidade grave. Esse indicador pode ser utilizado para se ter um conhecimento da gravidade dos defeitos abertos, podendo auxiliar os gestores de projetos a decidir por mudanças estratégicas para tratamento desses defeitos. Por exemplo, se o projeto estiver próximo a um *deadline* de entrega, todos os esforços devem ser concentrados na correção dos defeitos graves. Outra exemplo é quando os defeitos abertos de severidade grave ultrapassarem uma taxa pré-definida do total de defeitos abertos, os esforços devem ser concentrados na correção destes defeitos.

Quantidade de defeitos abertos por tipo

Considerando os tipos de testes existentes, baseados também nas características e sub-características citadas na ISO/IEC 9126 (2001), é possível classificar, por exemplo, testes nas seguintes categorias: funcionalidade, segurança, estabilidade, desempenho, interface. Caso um caso de teste falhe, gerando um defeito, é possível categorizar essas falhas de acordo com o tipo de caso de teste relacionado e gerar um indicador do panorama geral dos defeitos abertos do projeto conforme a figura a Fig. (23).

A depender do caráter do projeto, este indicador deverá ajudar o gestor do projeto na tomada de decisões. Supondo que o projeto em questão trata-se de um jogo com o foco educacional para crianças, é possível que em um software desse tipo seja prioritário corrigir defeitos de interface e desempenho em comparação a defeitos de segurança, por exemplo.

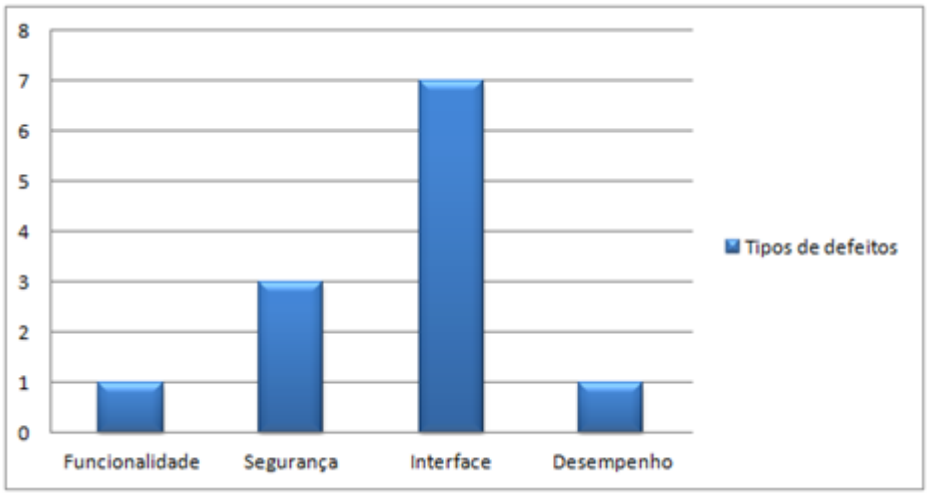

Figura 23 – Defeitos abertos por tipo

Já em software de caráter econômico ou que trate de dados sigilosos, possivelmente deve-se dar prioridade aos defeitos de segurança e funcionalidade.

Tipos de defeitos mais encontrados

Esse indicador assemelha-se ao anterior, entretanto, ao invés de indicar a quantidade de defeitos por tipo abertos num momento especifico, ele informa os tipos de defeitos mais encontrados desde o início do projeto, como se observa na Fig. (24).

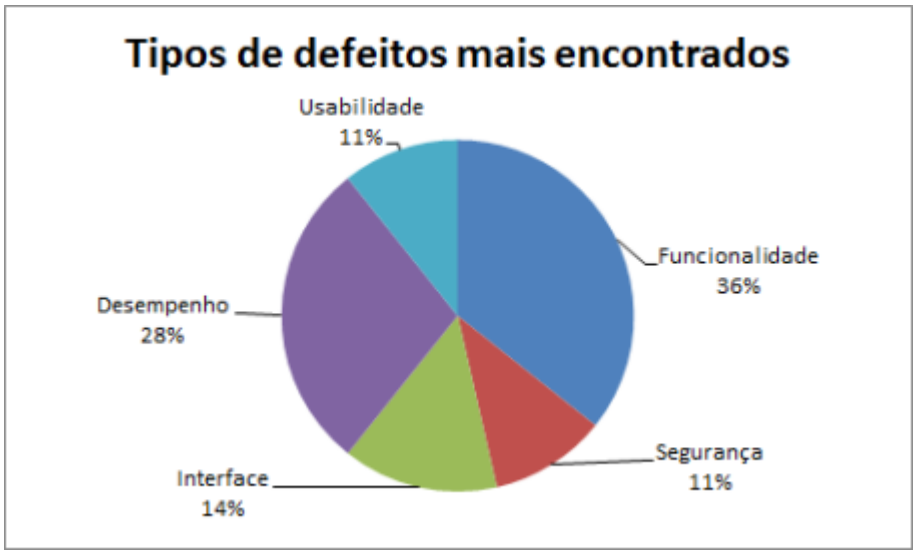

Figura 24 – Defeitos mais encontrados

Esse panorama geral da relação de defeitos mais encontrados pode ser visto para um projeto ou para vários projetos, mostrando ao gestor as fragilidades de seu processo ou equipe. Um alto número defeitos de funcionalidade podem indicar, dentre outras coisas, requisitos mal definidos. Um alto número defeitos de segurança podem indicar equipe pouco

treinada em conceitos e estratégias de segurança e um alto número defeitos de desempenho podem indicar também uma equipe com pouco conhecimento em desenvolvimento de código de alto desempenho. Tendo em vista esse tipo de informação, e considerando que cada projeto é desenvolvido por uma equipe diferente, o gestor pode, futuramente, equilibrar as equipes de acordo com o caráter do projeto, ou também, através de tais fragilidades, oferecer cursos para treinamento em determinadas áreas, a fim de aperfeiçoar a equipe de desenvolvimento, melhorando a qualidade dos produtos a serem desenvolvidos.

Cobertura de testes

Considerando um produto de software que contenha uma suíte de testes com 260 casos de testes definidos para um ciclo de teste, é possível ter um indicador da cobertura de testes dessa suíte de testes, conforme observado na Fig. (25).

Figura 25 – Cobertura de testes (executados)

O gráfico da Fig. (25) mostra o quanto da suíte de testes já foi executado. Entretanto, teste executado não significa teste executado e aprovado. É possível, então, obter um indicador especializado em que é indicada a quantidade de testes que foram executados e aprovados em relação aos que ainda não foram executados ou foram executados e não passaram.

Conforme é possível observar na Fig. (26), é possível entender não apenas a cobertura de testes, mas também a completude do produto desenvolvido até o momento, e consequentemente o andamento do projeto. Essa mesma visão de completude pode ser provida de acordo com o tempo, conforme é possível observar na Fig. (27).

O gráfico da Fig. (27) indica, em sua linha azul, a quantidade total de testes, e, em sua linha vermelha, os testes que a dado momento estão aprovados, e a evolução disso ao longo do tempo. Nessa simulação observa-se que, conforme esperado, inicialmente a

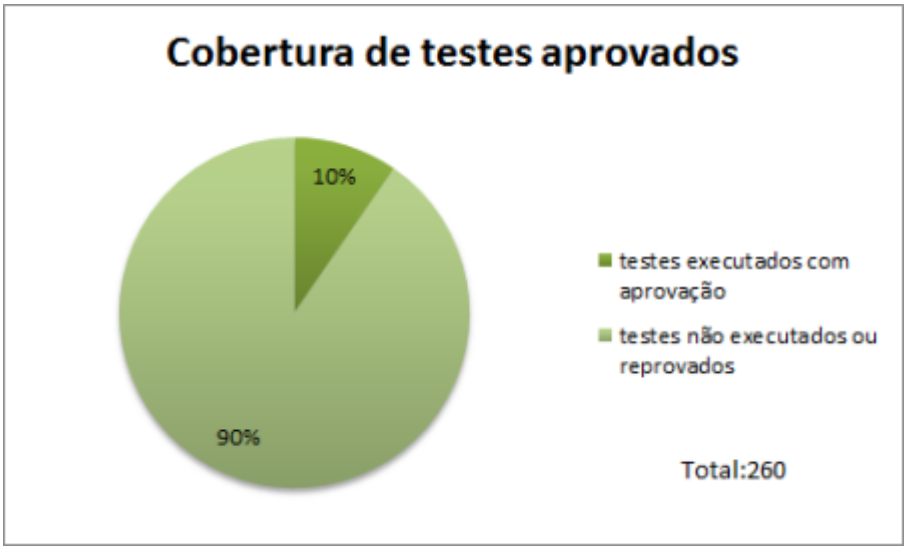

Figura 26 – Cobertura de testes (aprovados)

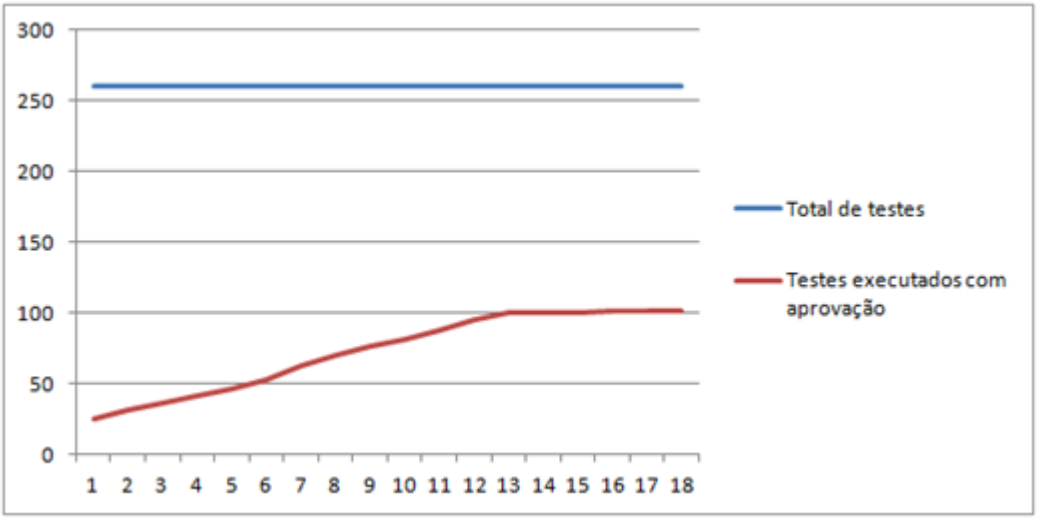

Figura 27 – Cobertura de testes aprovados ao longo do tempo

linha vermelha avança em encontro à linha azul. Contudo, em um dado momento ela para de crescer e se mantém estagnada. Isso pode indicar, por exemplo, que os testes pararam de ser executados ou que estão sendo executados, porém não estão obtendo os resultados esperados e as atividades de correção de defeitos não estão sendo eficazes.

Cobertura de testes por tipo

Similar ao indicador passado, esse indicador se especializa na cobertura por tipo de caso de teste. Esses tipos, conforme citados anteriormente, podem ser segurança, interface, e funcionalidade e podem ser utilizados em gráficos análogos ao dos indicadores de cobertura de teste, entretanto, podem especializar-se em um tipo específico, para que, caso o gestor tenha interesse em uma característica específica de qualidade, esses dados

estejam disponíveis de forma clara. Esse indicador será útil, aos gestores, em projetos que existam priorizações de características específicas de qualidade, ao prover uma visão de cobertura de testes aprovados da característica de qualidade solicitada, conforme é possível observar um exemplo na Fig. (28).

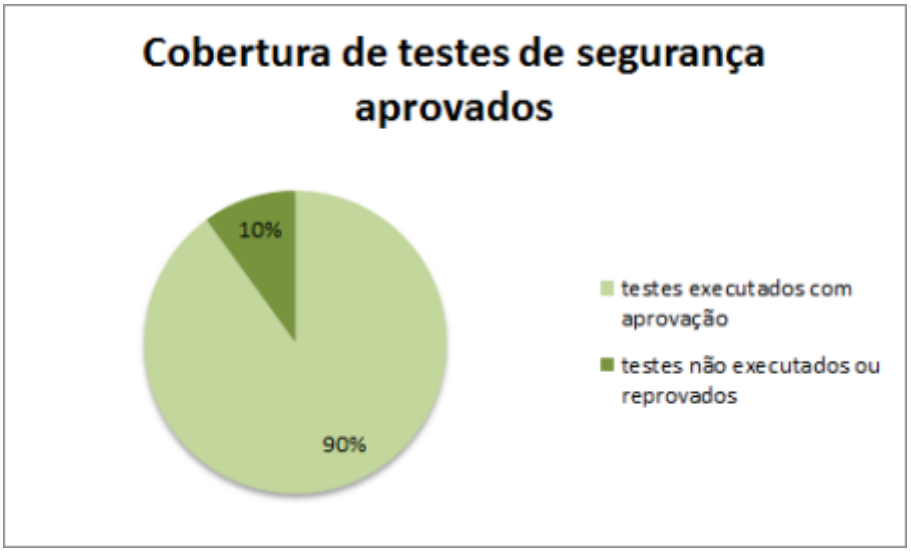

Figura 28 – Cobertura de testes por tipo

Cobertura de código

Esse indicador explicita o percentual de cobertura de código, em relação ao teste unitário, ao longo do tempo (Fig. (29)).

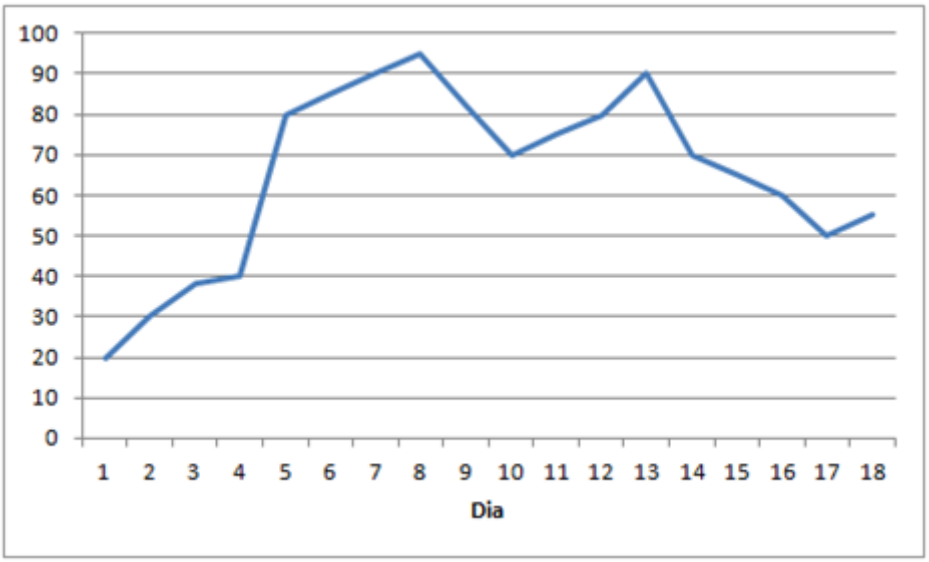

Figura 29 – Cobertura de código ao longo do tempo

Com esse indicador de cobertura de código o gestor do projeto pode acompanhar o andamento da cobertura de código ao longo do tempo. A cobertura pode diminuir

tendo em vista que o produto pode ainda estar sendo desenvolvido, portanto o código vem crescendo. Entretanto, dependendo da metodologia de desenvolvimento do projeto, pode ser que seja solicitado à equipe que o teste seja desenvolvido de acordo com o que o código também é desenvolvido. Com esse indicador o gestor pode acompanhar, ao longo do tempo, essa cobertura exigida.

É possível observar na Tab. (10) a relação entre os indicadores levantados e as questões definidas.

Questões	Indicadores
Que tipos de defeitos são mais encontrados?	- Quantidade de defeitos abertos por tipo - Tipos de defeitos mais encontrados
Qual gravidade dos defeitos encontrados?	- Densidade de defeitos abertos por severidade
Os defeitos estão sendo tratados e corrigidos?	- Total de defeitos encontrados VS Total de defeitos corrigidos - Total de defeitos graves encontrados VS Total de defeitos graves corrigidos - Quantidade de defeitos reabertos de acordo com o tempo
O produto foi testado suficientemente?	- Cobertura de testes - Cobertura de testes por tipo - Cobertura de código - Cobertura de testes aprovados
Como está a evolução da qualidade do software ao longo do tempo?	- Total de defeitos encontrados VS Total de defeitos corrigidos - Total de defeitos graves encontrados VS Total de defeitos graves corrigidos - Cobertura de testes - Cobertura de testes aprovados

Tabela 10 – Rastreabilidade entre questões e indicadores

Na Fig. (30) é possível observar a rastreabilidade entre Objetivos, Questões e Indicadores.

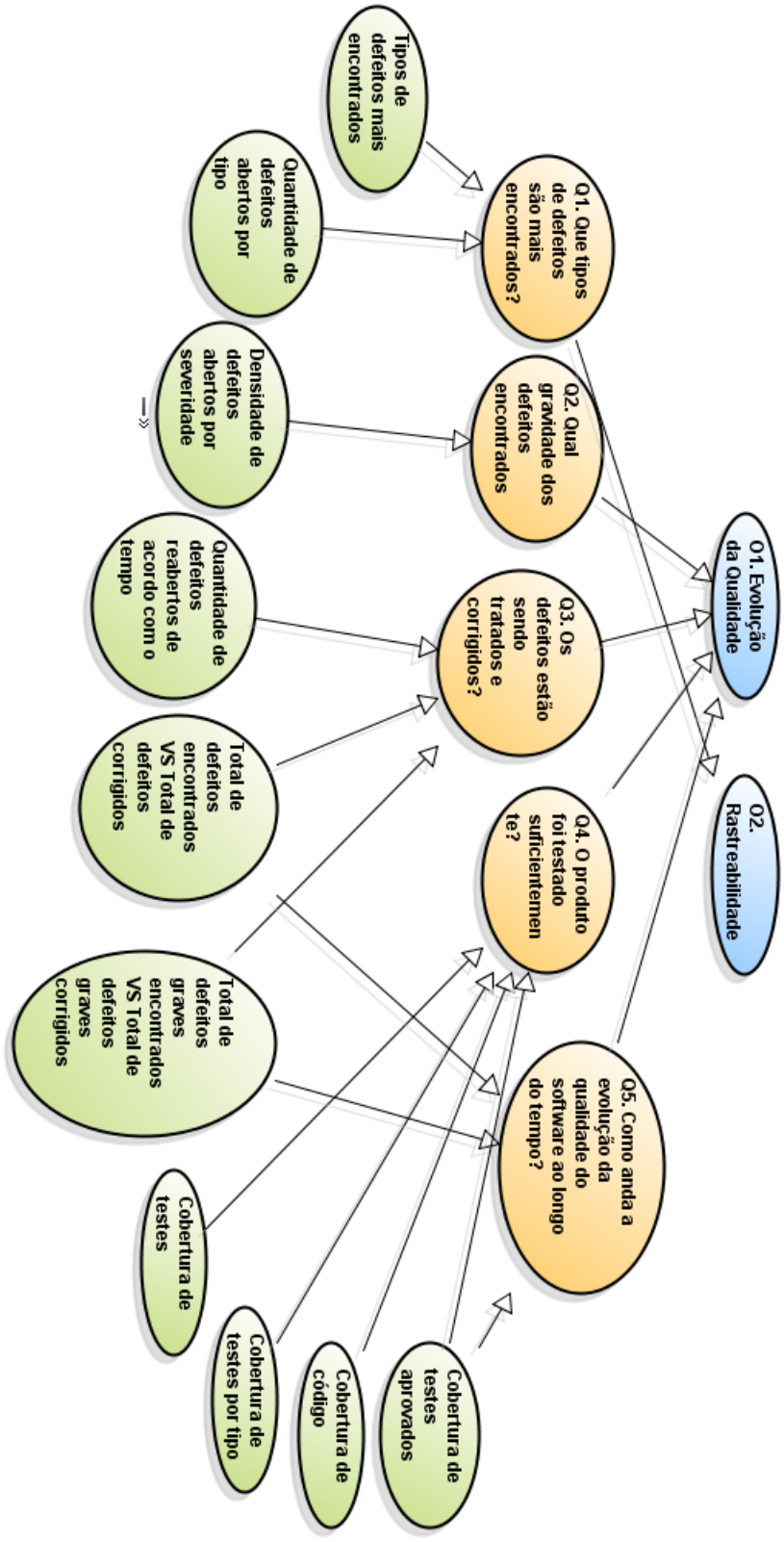

Figura 30 – Rastreabilidade Objetivos, Questões e Indicadores

5 O painel de acompanhamento de projetos: Protótipo

Este Capítulo destina-se à apresentação da ferramenta desenvolvida neste trabalho para a coleta e apresentação dos indicadores citados no Capítulo 4. Na Seção 5.1 serão abordados aspectos relacionados ao banco de dados das ferramentas de gestão de testes e defeitos, bem como a descrição da coleta dos dados necessários. Na Seção 5.2 será apresentado o painel desenvolvido.

5.1 Banco de dados

5.1.1 Estudo do banco de dados

A fim de viabilizar a coleta de dados para as medidas e indicadores a serem implantadas no painel de acompanhamento, torna-se necessário estudar o banco de dados das ferramentas citadas na subseção 4.2.2.

Ao total, as duas ferramentas contam com 108 tabelas no banco de dados. Para melhor entendimento e visualização do banco de dados se fazia necessário extrair o modelo relacional dessas tabelas. Entretanto, por se tratar de um número elevado de tabelas, foi realizada uma análise de cada uma dessas tabelas a fim de investigar quais, num primeiro momento, seriam ou não úteis a este trabalho. Algumas tabelas como: attachments, attach_data, email_setting, keyworddefs, test_attachments, dentre outras contiam informação não relevantes aos indicadores definidos, foram desconsideradas na geração do modelo, que pode ser conferido na figura 31, que contém um total de 20 tabelas.

Foi realizado então um estudo e descrição de cada uma das tabelas:

- bug_severity: Define as severidades que podem ser associados a um defeito. Exemplos: Crítico, Médio, Melhoria.
- bug_status: Define os estados que um defeito pode assumir. Exemplos: Aberto, rejeitado, resolvido, em análise, em progresso.
- bugs: Define as características gerais contidas em um defeito. Contém informações como: nome, data de criação, descrição, prioridade, produto o qual o defeito está associado, dentre outras.

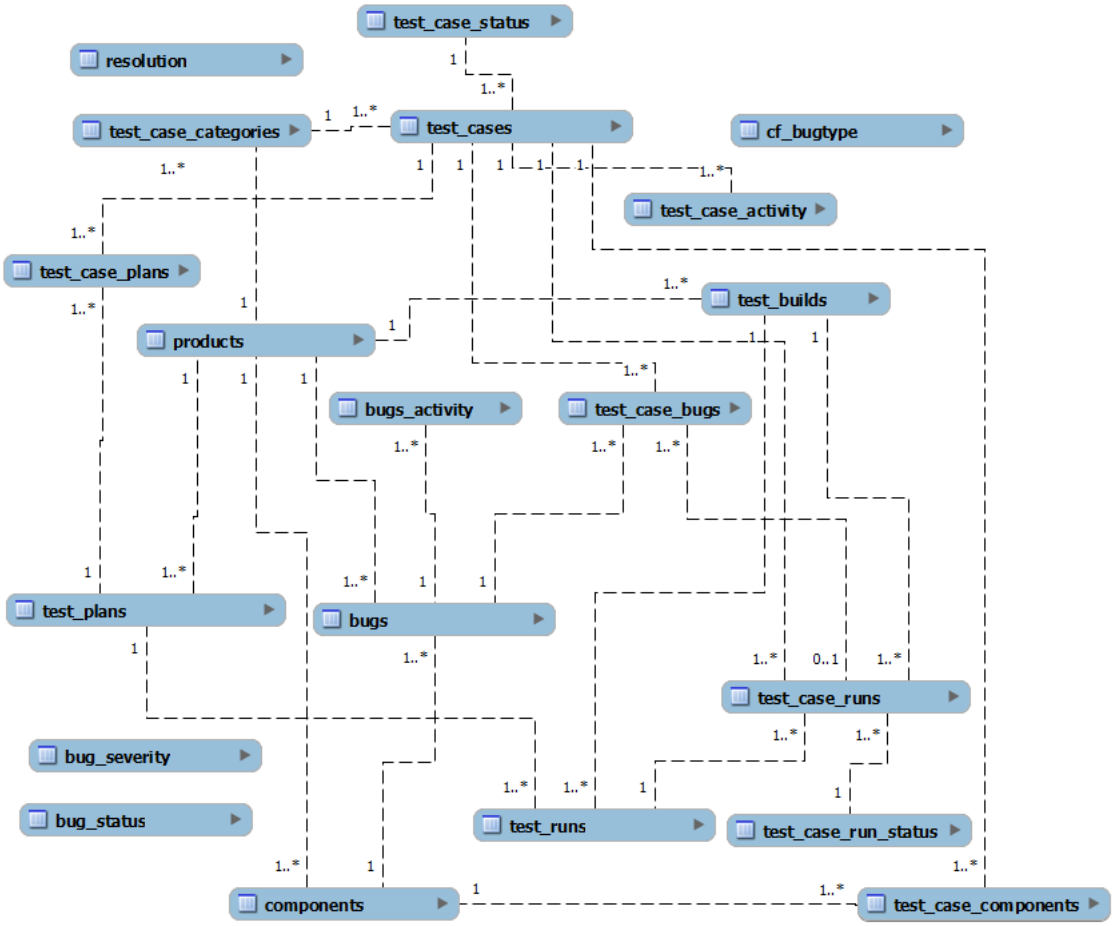

Figura 31 – Modelo Relacional do Banco de Dados

- bugs_activity: Define características a serem salvas sempre que há alguma mudança no defeito, como quem mudou, quando mudou, que campo foi mudado, qual estado anterior, qual estado atual, dentre outras.
- cf_bugtype: Define os tipos de defeito que existem. Exemplos: funcionalidade, desempenho, segurança, interface.
- components: Define os componentes que compõe um produto.
- products: Define os produtos aos quais os defeitos, planos de testes, dentre outras coisas, estão associados.
- resolution: Define os tipos de resolução que podem ser atribuídos ao bug. Exemplo: corrigido, duplicado, inválido.
- test_builds: Define as builds, ou seja, identificação de compilações de cada produto que poderao ser testadas.
- test_case_activity: Define algumas informações que serão salvas quando há mudanças em casos de teste.

- test_case_bugs: Relaciona o bug ao caso de teste a qual está associado em qual test_run ele foi identificado.
- test_case_categories: Define as categorias dos casos de teste. Exemplo: funcionalidade, segurança.
- test_case_components: Relaciona cada caso de teste a um componente do product.
- test_case_plans: Relaciona os casos de testes com os planos de testes aos quais pertencem.
- test_case_run_status: Define os estados que um caso de teste pode assumir durante sua execução. Exemplos: Em espera, passou, falhou, pausado.
- test_case_runs: Relaciona os casos de teste (e seus estados) com a run.
- test_case_status: Define os estados de um caso de teste fora de execução: proposto, confirmado, desabilitado.
- test_cases: Define os casos de testes, relacionando a cada um deles prioridade, autor, categoria, dentre outros.
- test_plans: Define características do plano de teste como nome, data de criação, autor, e relaciona o plano de teste com seu tipo e produto associado.
- test_runs: Define características como nome, data e versão e associa com qual plano de testes a run corresponde, qual gerente de execução, dentre outras.

5.1.2 Mapeamento entre medidas e tabelas

Após esse estudo foi realizado um mapeamento entre as medidas básicas e as tabelas levantadas, conforme pode ser observado na Tab. 11.

5.1.3 Extraindo as informações necessárias para os indicadores

O próximo passo realizado é a extração das informações dos bancos de dados para a construção dos indicadores citados na Seção 4.3.3.3.

Total de defeitos encontrados VS Total de defeitos corrigidos

Para a construção desse indicador, é necessário que se tenha uma lista com todos os defeitos, contendo as seguintes informações: se o defeito está aberto ou não; data que o defeito foi encontrado; data de última modificação. Estas informações estão contidas nas tabelas bugs e bug_status, nos campos bug_status.is_open, bugs.creation_ts e bugs.delta_ts, respectivamente.

Medidas Básicas	Descrição	Tabelas
Quantidade de Defeitos	Quantidade de defeitos encontrados nas atividades de teste.	bugs
Severidade de Defeitos	Classifica o defeito de acordo com sua severidade.	bug_severity
Tipo de Defeitos	Classifica o defeito de acordo com o tipo.	cf_bugtype
Estado de Defeitos	Classifica o defeito de acordo com o seu estado.	bug_status, resolution, bugs_activity
Estado do caso de teste	Classifica o caso de teste de acordo com seu estado atual.	test_case_run_status, test_cases
Tipos de casos de teste	Classifica o caso de teste de acordo com seu tipo.	test_case_categories
Quantidade de casos de teste	Quantidade de casos de testes que existe para um produto	test_cases
Quantidade de ciclos de teste	Quantidade de ciclos de teste executado para um produto	test_runs, test_builds

Tabela 11 – Relação entre medidas básicas e tabelas do banco de dados

A seguir é possível observar a query SQL escrita para extrair tais informações:

```
SELECT bug_status.is_open, bugs.creation_ts, bugs.
    delta_ts
WHERE bugs,  bug_status
AND bugs.bug_status = bug_status.value
AND bugs.product_id = 1
ORDER BY bugs.creation_ts ASC;
```

O uso do product_id = 1 é apenas para exemplificar qual produto está sendo feita a análise. Essa informação deverá vir da interface da aplicação de acordo com qual produto o usuário queira consultar. A ordenação crescente pela data de criação foi usada para facilitar o manuseio dos dados no momento da construção do gráfico.

Total de defeitos graves encontrados VS Total de defeitos graves corrigidos

Para a construção desse indicador, é necessário que se tenha informações bem similares ao do indicador passado. A query é bastante similar, entretanto conta com uma nova cláusula para que sejam obtidos apenas os defeitos graves. A seguir e possível observar a query.

```
SELECT bug_status.is_open, bugs.creation_ts, bugs.
    delta_ts
FROM bugs, bug_status
WHERE bugs.bug_status = bug_status.value
```

```
AND bugs.product_id = 1
AND bugs.bug_severity = 'critical'
ORDER BY bugs.creation_ts ASC
```

Quantidade de defeitos reabertos de acordo com o tempo

Para a construção desse indicador, é necessária uma lista não de todos os defeitos reabertos, mas de todas as ocorrências de reabertura de defeitos de acordo com o tempo, pois um mesmo defeito pode ter sido reaberto várias vezes. Com isso é necessário que se tenha essa lista de ocorrências com a informação de quando cada ocorrência foi detectada. Esta informação está contida na tabela bugs_activity, no campo bug_when.

A seguir e possível observar a query SQL escrita para extrair tais informações:

```
SELECT bugs_activity.bug_when, bugs.creation_ts
FROM bugs, bugs_activity
WHERE bugs_activity.bug_id = bugs.bug_id
AND bugs.product_id = 2
AND bugs_activity.added = 'REOPENED'
ORDER BY bugs.creation_ts ASC
```

O uso do creation_ts, de quando o defeito foi encontrado, é colocada apenas como informação extra caso seja necessário um parâmetro para data de inicial do grafico. Mais uma vez, o uso do product_id = 2 é apenas para exemplificar.

Defeitos abertos por severidade

Para a construção desse indicador, é necessário uma lista com a severidade de cada defeito aberto. Essas informações estão nas tabelas bugs e bug_status. A severidade do defeito pode ser obtida através do campo bugs.bug_severity e a informação se o defeito está ou não aberto através do bug_status.is_open, que quando igual a 1, quer dizer que o defeito está aberto.

A seguir é possível observar a query escrita para extrair tais informações:

```
SELECT bugs.bug_severity
FROM bugs, bug_status
WHERE bugs.bug_status = bug_status.value
AND bug_status.is_open = 1
AND bugs.product_id = 2
```

Quantidade de defeitos abertos por tipo

Para a construção desse indicador, é necessário uma lista com o tipo de defeito de cada defeito aberto. O tipo de cada defeito pode ser encontrado na tabela bugs, através do campo bugs.cf_bugtype. E para saber se um defeito está aberto é necessária a tabela bug_status, através do campo bug_status.is_open.

```
SELECT bugs.cf_bugtype
FROM bugs, bug_status
WHERE bugs.bug_status = bug_status.value
AND bug_status.is_open = 1
AND bugs.product_id = 2
```

Tipos de defeitos mais encontrados

Para a construção desse indicador, é necessária uma lista com o tipo de defeito de cada defeito encontrado. É similar ao passado, entretanto sem a cláusula que filtra apenas os defeitos abertos. Para este indicador, é necessário listar o campo bugs.cf_bugtype da a tabela bugs.

```
SELECT bugs.cf_bugtype
FROM bugs
WHERE bugs.product_id = 2
```

Outra possibilidade para esse indicador é não o restringir por produto, mas sim por todos os produtos. Para isso retira-se a cláusula WHERE.

Cobertura de testes

Para esse indicador, é necessário uma lista com o estado de cada caso de teste em um ciclo de testes. Para coletar tais informações, são necessárias as tabelas test_cases e test_case_runs. O estado de cada caso de teste está no campo test_case_runs.case_run_status_id, onde caso assuma os valores 2 ou 3, (passar ou falhar, respectivamente), significa que foram executados.

```
SELECT test_case_runs.case_run_status_id
FROM test_case_runs, test_cases
WHERE test_case_runs.build_id = 4
AND test_cases.case_id = test_case_runs.case_id
```

Essa mesma query serve de insumo para a especialização citada desse indicador, que mostra o gráfico de cobertura de testes aprovados, e não simplesmente executados. Para isso consdera-se como cobertura apenas os casos onde o valor do campo test_case_runs.case_run_status_id é igual a 2.

O uso do test_case_runs.build_id = 4 é para apenas exemplificar qual ciclo de testes está sendo analisado, normalmente o ultimo. A última build, pode ser extraída através da seguinte query:

```
SELECT test_builds.build_id
FROM test_builds
WHERE test_builds.product_id = 2
ORDER BY test_builds.build_id DESC
```

Esta query poderá retornar para a aplicação uma lista de builds a qual a primeira build listada e a última. E o valor test_builds.build_id pode ser passado para a query de cobertura de testes.

Cobertura de testes aprovados por tipo

Este indicador é similar à especialização do indicador passado, entretanto ele ainda recebe um filtro do tipo de caso de teste que deseja saber a cobertura. Para é utilizada também a tabela test_case_categories e os campos test_case_categories.name e case_run_status_id.

```
SELECT test_case_runs.case_run_status_id,
    test_case_categories.name
FROM test_case_runs, test_case_categories, test_cases
WHERE test_case_runs.build_id = 4
AND test_cases.case_id = test_case_runs.case_id
AND test_cases.category_id = test_case_categories.
    category_id
AND test_cases.category_id = 2
```

Cobertura de testes aprovados ao longo do tempo

Para a construção deste indicador, são necessárias a quantidade total de casos de testes e a data de de término da execução de cada caso de teste, que é obtida através do campo close_date da tabela test_case_runs. É necessário também saber o estado de cada caso de teste através do campo case_run_status_id.

```
SELECT test_case_runs.case_run_status_id, close_date
FROM test_case_runs, test_cases
WHERE test_case_runs.build_id = 2
AND test_cases.case_id = test_case_runs.case_id
ORDER BY test_case_runs.close_date ASC
```

5.2 Protótipo

5.2.1 Tecnologia utilizada

Para desenvolvimento da aplicação, foi utilizado um *framework* de desenvolvimento em PHP, o CodeIgniter [1]. Esse *framework* foi desenvolvido sob o padrão de arquitetura de *software* MVC (*Model-View-Controller*), que separa as camadas lógicas e de negocio da camada de apresentação. Os *models*, ou modelos, são os componentes que fazem parte da camada de abstração de dados. Normalmente são responsáveis por gravar ou recuperar dados do banco de dados. São nas *models* que as *queries* da mostradas na Sec. 5.1 ficarão. As *views* são a parte principal da camada de apresentação. São as *views* que recebem os dados dos *controllers* e não deverão se comunicar diretamente com os *models* ou com o banco de dados. Os *controllers* são a camada responsável pela comunicação entre os *models* e os *views* (UPTON; ELLIS; ALLARD, 2007). Através da Fig. 32 é possível observar de maneira mais clara como se dá esse fluxo de dados.

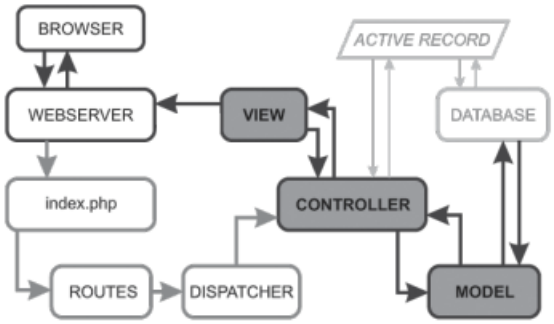

Figura 32 – Fluxo de dados MVC com CodeIgniter (GABARDO, 2012)

Abstraindo alguns módulos da Fig. (32) basicamente o funcionamento pode ser observado na Fig. (33), onde o *controller* em indicador.php ao receber uma requisição da função grafico_aberto_severidade, por exemplo, recebendo como parâmetro o id do produto, acessa o *model* em bug.php. Por sua vez o *model* obtem do banco de dados as informações necessárias e retorna para o controlador, que assim carrega uma *view* correspondente, e envia o resultado formatado pela *view* para exibição no *browser*.

[1] CodeIgniter: http://ellislab.com/codeigniter/

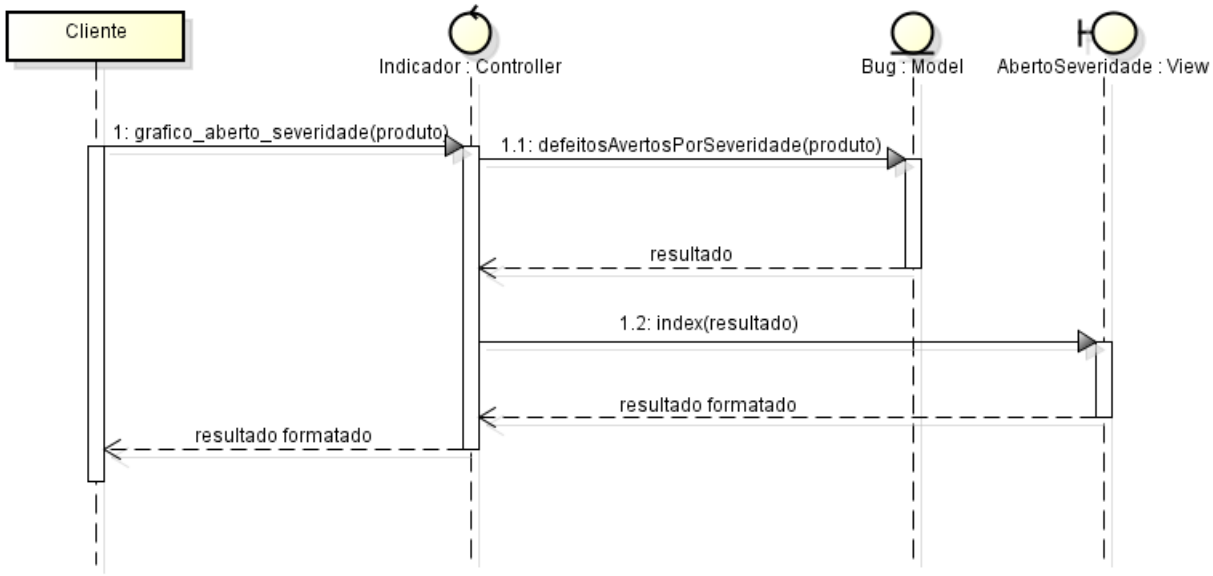

Figura 33 – Fluxo de dados MVC com CodeIgniter

A interface gráfica foi construída com o auxílio do framework Twitter Bootstrap [2] e os graficos com o o jqPlot [3].

5.2.2 Protótipo

Cada indicador pode ser selecionado em um menu na ferramenta conforme demonstrado na Fig. (34). Ainda na mesma figura é possível observar um menu para a seleção do produto que se deseja. O código-fonte pode ser encontrado no repositório https://bitbucket.org/adrianobarboza/tcc/.

[2] Twitter Bootstrap: http://getbootstrap.com/
[3] jqPlot: http://www.jqplot.com/

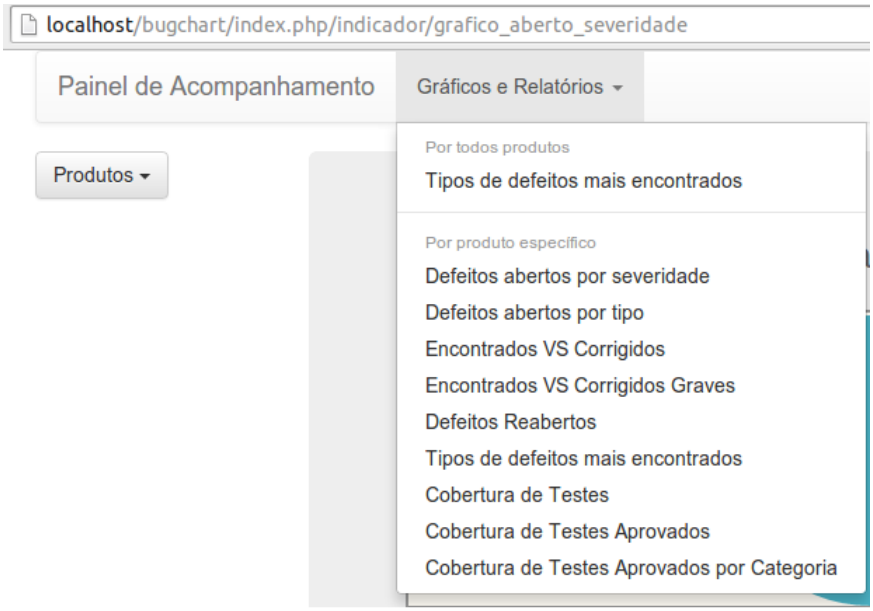

Figura 34 – Menus de seleção de indicador e produto

O indicador "Total de defeitos encontrados VS Total de defeitos corrigidos" pode ser observado na Fig. (35).

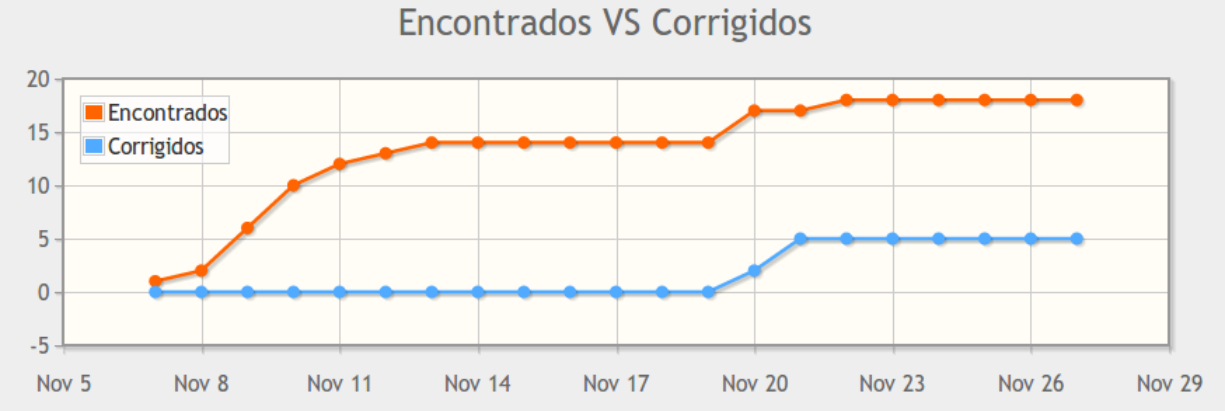

Figura 35 – Defeitos encontrados vs defeitos corrigidos

O indicador "Total de defeitos graves encontrados VS Total de defeitos graves corrigidos", que se trata do indicador passado com um filtro de defeitos graves, pode ser observado na Fig. (36).

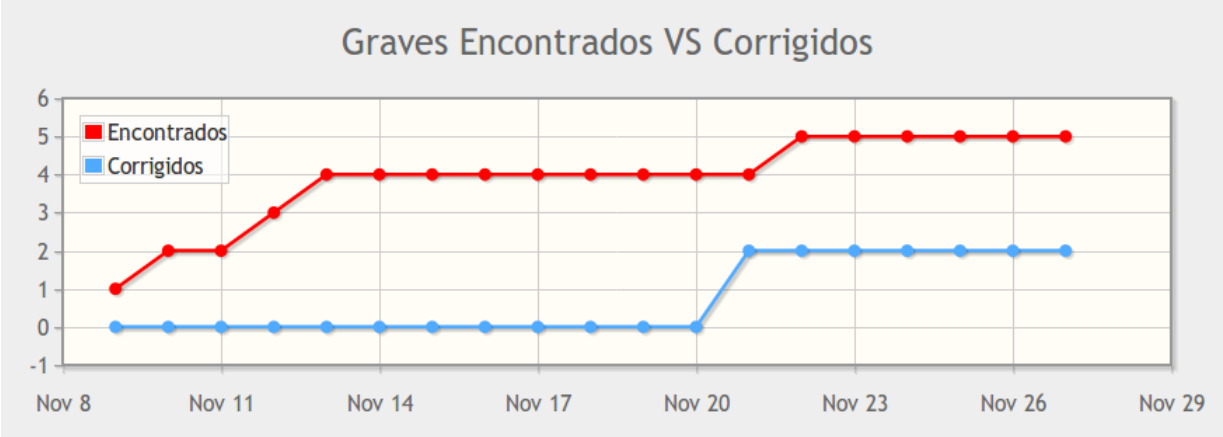

Figura 36 – Defeitos graves encontrados vs defeitos graves corrigidos

O indicador "Quantidade de defeitos reabertos de acordo com o tempo" pode ser observado na Fig. (37).

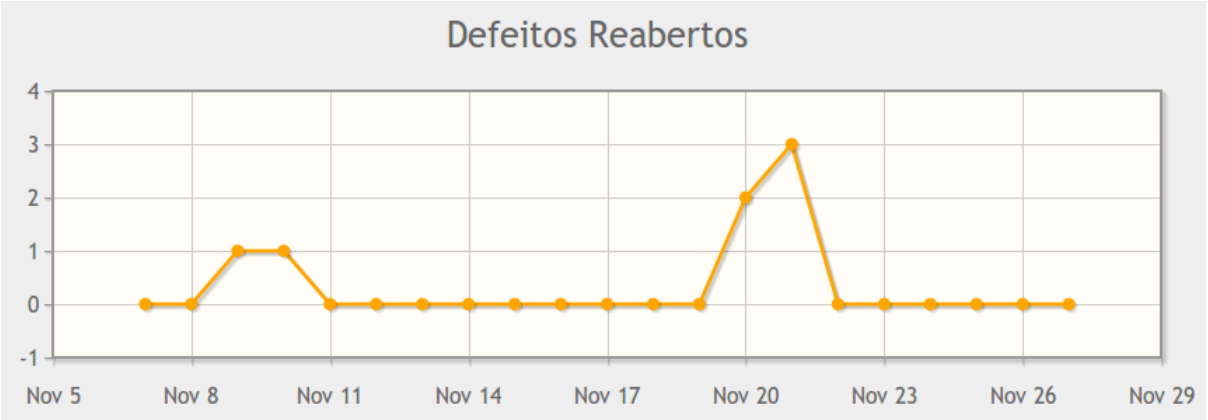

Figura 37 – Quantidade de defeitos reabertos de acordo com o tempo

O indicador "Defeitos abertos por severidade" pode ser observado na Fig. (38).

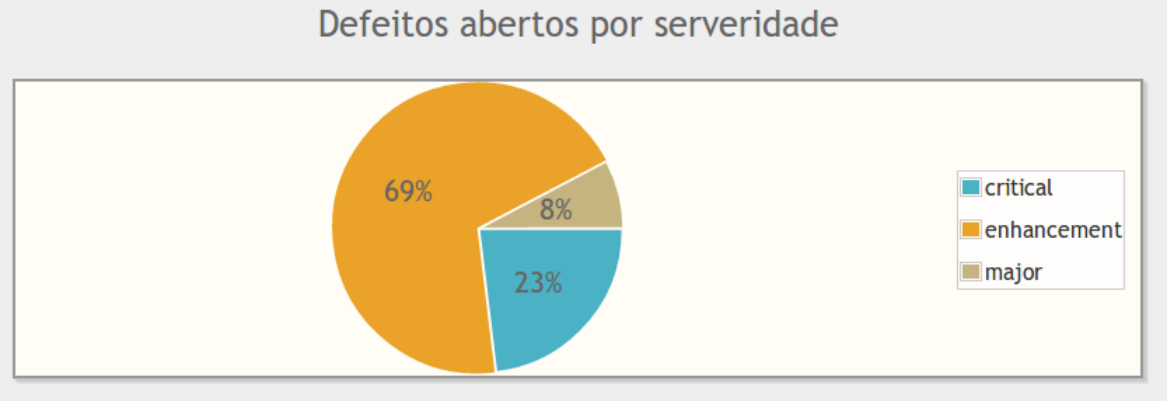

Figura 38 – Defeitos abertos por severidade

O indicador "Quantidade de defeitos abertos por tipo" pode ser observado na Fig. (39).

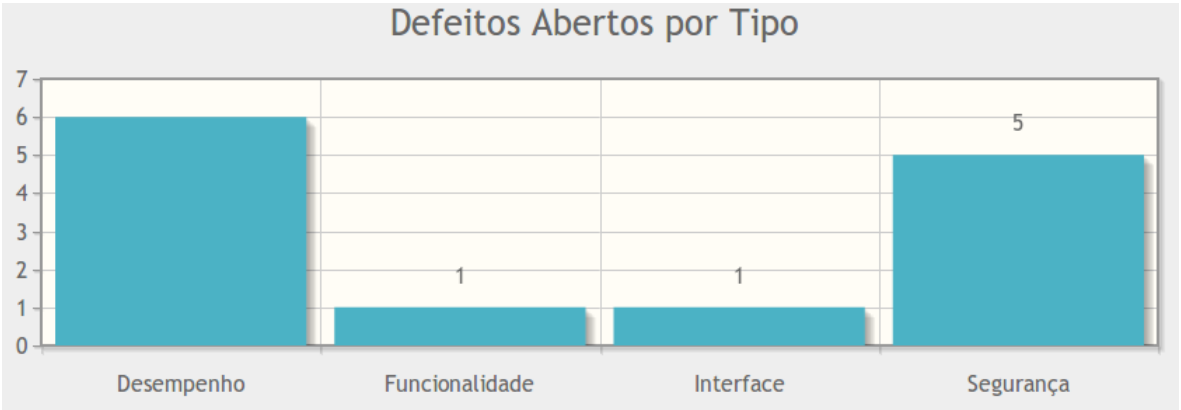

Figura 39 – Tipos de defeitos mais encontrados por produto

O indicador "Tipos de defeitos mais encontrados em todos os produtos" pode ser observado na Fig. (40), e sua versão por produto específico pode ser observada na Fig. (41).

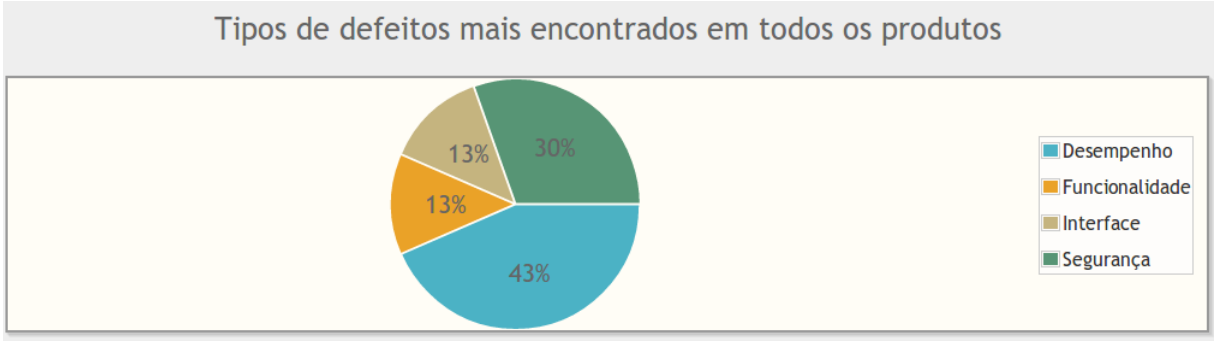

Figura 40 – Tipos de defeitos mais encontrados em todos os produtos

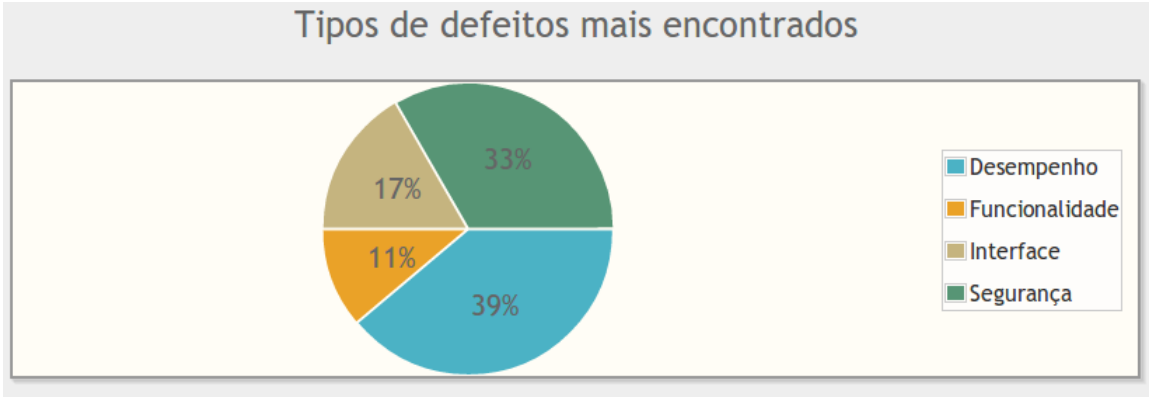

Figura 41 – Tipos de defeitos mais encontrados por produto

O indicador "Cobertura de testes" pode ser observado na Fig. (42), e sua versão de cobertura de testes aprovados na Fig. (43).

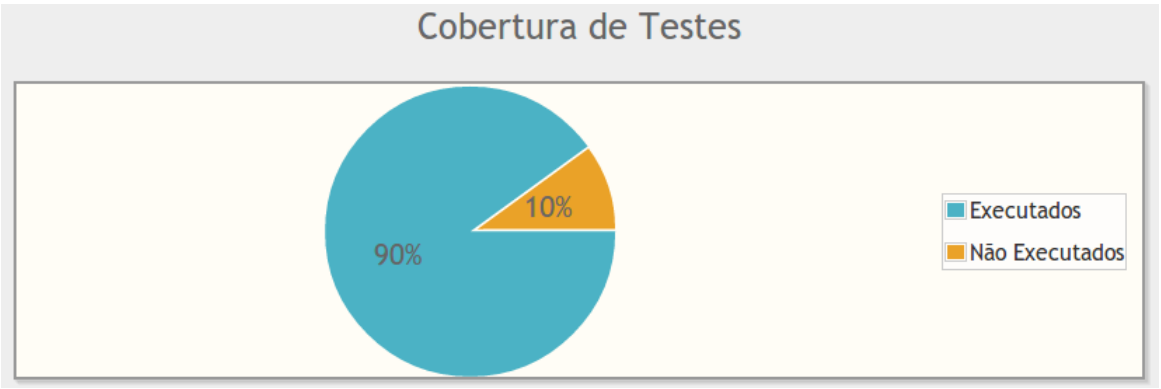

Figura 42 – Cobertura de testes

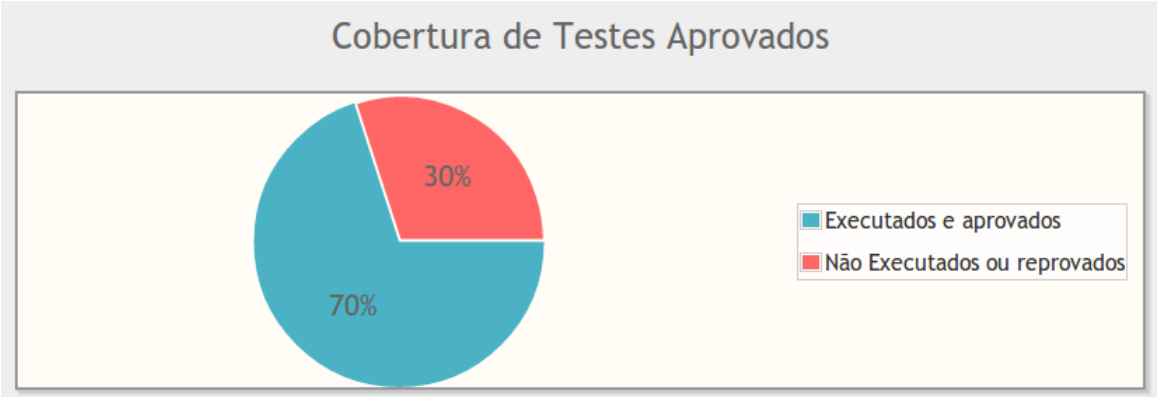

Figura 43 – Cobertura de testes aprovados

O indicador "Cobertura de testes aprovados por tipo" pode ser observado na Fig. (44).

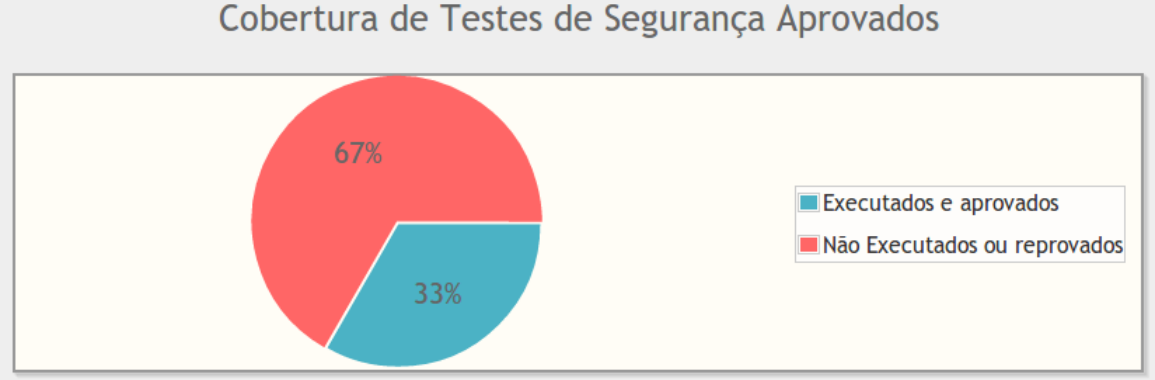

Figura 44 – Cobertura de testes aprovados por tipo

E, por último, o indicador "Cobertura de testes aprovados ao longo do tempo" pode ser observado na Fig. (45).

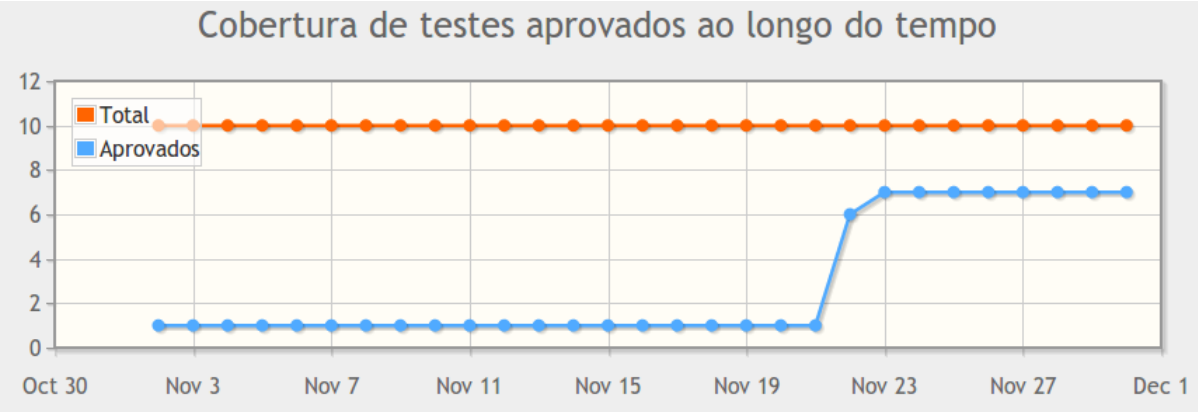

Figura 45 – Cobertura de testes aprovados ao longo do tempo

6 Considerações Finais

Por meio da revisão bibliográfica, foi possível salientar a importância da Qualidade de Software no cenário atual, e a sua consequente importância no contexto de gerenciamento de projetos. As atividades de Teste de Software, inseridas no contexto de Gestão da Qualidade, apresentam-se como indispensáveis no ciclo de vida de desenvolvimento e manutenção de software. Dadas as dificuldades que gestores de projeto enfrentam em suas atividades, as medidas e indicadores podem representar importantes aliados no auxílio a esses gestores a terem visibilidade sobre o estado do produto e no processo de tomada de decisões.

Antes de ser iniciado o processo de desenvolvimento do painel de acompanhamento o qual este trabalho se propõe a desenvolver, fez-se necessário o levantamento de medidas e indicadores a serem implementados no painel. Contudo, essas medidas e indicadores devem ser coerentes com o contexto ao qual estarão inseridos, atendendo suas necessidades de informação. Portanto, foram realizadas atividades que comportavam o fluxo de definição de medidas proposto pela abordagem GQM, na definição de objetivos, questões, medidas e indicadores alinhados.

Uma etapa crítica do trabalho foi o estudo do banco de dados das ferramentas de gestao de testes e defeitos. Contudo, o banco de dados foi entendido e constatou-se aonde estavam as informações necessárias a serem coletadas. Com isso, foi possível construir as *queries* para coletas dessas informações, que foram repassadas à aplicação desenvolvida para mostrá-las na forma dos indicadores definidos previamente. Com isso, com exceção do indicador de cobertura de código, dado que no ambiente não se trabalha com o código dos produtos testados, todos os indicadores foram implementados.

Referências

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Wiley, 1994. Citado 4 vezes nas páginas 15, 29, 30 e 31.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. Goal question metric paradigm. In: *Encyclopedia of Software Engineering*. [S.l.]: John Wiley & Sons, 1994, (2 Volume Set). Citado na página 29.

BASILI, V. R. et al. Lessons learned from 25 years of process improvement. In: . ACM Press, 2002. p. 69. ISBN 158113472X. Disponível em: <http://portal.acm.org/citation-.cfm?doid=581339.581351>. Citado na página 29.

BOEHM, B. W. *Guidelines for Verifying and Validating Software Requirements and Design*. 1979. Disponível em: <http://csse.usc.edu/csse/TECHRPTS/1979/usccse79-501/usccse79-501.pdf>. Citado na página 26.

DASKALANTONAKIS, M. A practical view of software measurement and implementation experiences within motorola. *IEEE Transactions on Software Engineering*, 1992. v. 18, n. 11, p. 998–1010, nov. 1992. ISSN 00985589. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=177369>. Citado na página 29.

DeMarco, T. *Controlling Software Projects: Management, Measurement, and Estimates*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1986. ISBN 0131717111. Citado na página 28.

FENTON, N. E.; PFLEEGER, S. L. *Software metrics: a rigorous and practical approach*. Boston: PWS Pub., 1997. ISBN 0534954251 9780534954253. Citado 2 vezes nas páginas 28 e 29.

GABARDO, A. *Php E Mvc Com Codeigniter*. NOVATEC, 2012. ISBN 9788575223338. Disponível em: <http://books.google.com.br/books?id=M6DANAEACAAJ>. Citado 2 vezes nas páginas 16 e 66.

GRADY, R. Successfully applying software metrics. *Computer*, 1994. v. 27, n. 9, p. 18–25, set. 1994. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org-/lpdocs/epic03/wrapper.htm?arnumber=312034>. Citado 3 vezes nas páginas 15, 31 e 32.

HAZAN, C.; LEITE, J. C. S. do P. Indicadores para a gerência de requisitos. In: *WER'03*. [S.l.: s.n.], 2003. p. 285–301. Citado na página 28.

HUTCHESON, M. L. *Software testing fundamentals: methods and metrics*. Indianapolis, Ind.: Wiley Pub., 2003. ISBN 047143020X 9780471430209. Citado 4 vezes nas páginas 15, 31, 32 e 33.

ISO/IEC 15939. *ISO/IEC 15939. Systems and software engineering - Measurement process*. [S.l.]: ISO/IEC, 2007. Citado 2 vezes nas páginas 28 e 30.

ISO/IEC 9126. *ISO/IEC 9126. Software engineering - Product quality.* [S.l.]: ISO/IEC, 2001. Citado 2 vezes nas páginas 25 e 51.

LEAL, K. A. B. *Relato de experiêcia da implantação de boas práticas de Engenharia de Software em um ambiente heterogêneo.* Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, 2008. Citado 2 vezes nas páginas 24 e 25.

LEWIS, W. E. *Software testing and continuous quality improvement.* Boca Raton: Auerbach, 2000. ISBN 0849398339. Citado 3 vezes nas páginas 25, 26 e 27.

McGarry, J. *Practical software measurement: objective information for decision makers.* Boston, MA: Addison-Wesley, 2002. ISBN 0201715163. Citado na página 28.

MONTEIRO, L. F. S. *Definição de um Catálogo de Medidas para a Análise de Desempenho de Processo de Software.* 2008. Citado 5 vezes nas páginas 17, 21, 29, 30 e 31.

NAIR, T. G.; SUMA, V.; TIWARI, P. K. Significance of depth of inspection and inspection performance metrics for consistent defect management in software industry. *IET Software*, 2012. v. 6, n. 6, p. 524, 2012. ISSN 17518806. Disponível em: <http://digital-library.theiet.org/content/journals/10.1049/iet-sen.2011.0148>. Citado na página 25.

NBR/ISO 9000. *NBR ISO 9000. Sistemas de gestão da qualidade - Fundamentos e vocabulário.* [S.l.]: NBR/ISO, 2005. Citado na página 25.

PAUL, R. et al. Software metrics knowledge and databases for project management. *IEEE Transactions on Knowledge and Data Engineering*, 1999. v. 11, n. 1, p. 255–264, 1999. ISSN 10414347. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=755633>. Citado na página 24.

PAVUR, R.; JAYAKUMAR, M.; CLAYTON, H. Software testing metrics: Do they have merit? *Value in Health*, 2000. v. 3, n. 1, p. 5–10, 2000. ISSN 10983015 (ISSN). Citado na página 28.

PMBOK. *Um guia do conhecimento em gerenciamento de projectos (Guia PMBOK).* Newtown Square, Pa.: Project Management Institute, 2008. ISBN 9781933890708 1933890703. Citado 2 vezes nas páginas 23 e 24.

PRESSMAN, R. S. *Engenharia de software.* São Paulo: McGraw-Hill, 2006. ISBN 8586804576 9788586804571. Citado 3 vezes nas páginas 26, 27 e 28.

SHERRIFF, M. Utilizing verification and validation certificates to estimate software defect density. In: . ACM Press, 2005. p. 381. ISBN 1595930140. Disponível em: <http://portal.acm.org/citation.cfm?doid=1081706.1081768>. Citado na página 21.

SOMMERVILLE, I. *Software engineering.* Boston: Pearson, 2011. ISBN 9780137035151 0137035152 0137053460 9780137053469. Citado 5 vezes nas páginas 21, 23, 24, 25 e 27.

SOUSA, E. J.; MARINHO, D. S. *Pesquisa de Qualidade no Setor de Software Brasileiro.* [S.l.], 2009. Citado na página 25.

STARK, G.; DURST, R.; VOWELL, C. Using metrics in management decision making. *Computer*, 1994. v. 27, n. 9, p. 42–48, set. 1994. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=312037>. Citado na página 28.

TAKARA, A.; BETTIN, A. X.; TOLEDO, C. M. T. Problems and pitfalls in a CMMI level 3 to level 4 migration process. In: . IEEE, 2007. p. 91–99. ISBN 0-7695-2948-8, 978-0-7695-2948-6. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper-.htm?arnumber=4335237>. Citado na página 29.

TRODO, L. D. *Uso de métricas nos testes de software*. Tese (Trabalho de conclusão de graduação) — Universidade Federal do Rio Grande do Sul. Instituto de Informática, 2009. Citado 2 vezes nas páginas 31 e 32.

UPTON, D.; ELLIS, R.; ALLARD, D. *CodeIgniter for rapid PHP application development improve your PHP coding productivity with the free compact open-source MVC CodeIgniter framework!* Birmingham, U.K.: Packt Pub., 2007. ISBN 9781847191755 1847191754. Disponível em: <http://www.books24x7.com/marc.asp?bookid=30380>. Citado na página 66.

WEBER, T. S. *Tolerância a falhas - conceitos e exemplos*. [S.l.], 2003. Citado na página 26.